Chapter 3: Sequential Logic Systems

1. The $\overline{S}$-$\overline{R}$ Latch

Learning Objectives:

At the end of this topic you should be able to:

- design a Set-Reset latch based on NAND gates;
- complete a sequential truth table to describe its action.

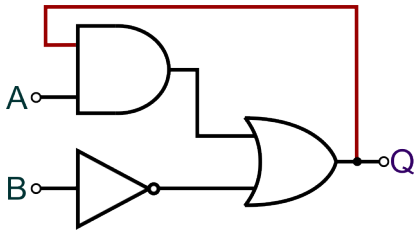
Introduction

The logic systems investigated so far have been combinational logic systems. The output of these systems depends only on the current state of the inputs and not on the sequence that led up to that state. As soon as inputs are changed, any information about the previous state of the inputs is lost. Combinational logic systems have no 'memory'.

In a sequential logic system, the output also depends on past input states. However, a sequential system has some form of 'memory' of what happened previously.

A sequential logic system uses feedback to allow the current output state to influence future input states.

For example, the logic system opposite has a feedback loop, shown in red, between the final output **Q** and one input of the AND gate. Only when **Q** is logic **1** can input **A** have any effect on the system.

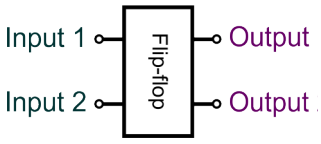


Only if the present state of the system is known can we predict future behaviour. For that reason, it is called a sequential system – events occur in a particular sequence.

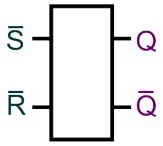
Flip-flops

The basic building block of a sequential system is a sub-system called a 'flip-flop' (or 'bistable'.) It is so called because it has two stable states (hence '**bi**stable') and it flips (or flops) between the two.

It is a sub-system with two input terminals and two output terminals, usually.



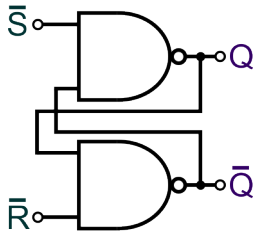
There are a number of different types of flip-flop.The simplest is probably the Set-Reset Latch (or **$\overline{S}$-$\overline{R}$** latch) shown opposite:



Notes:

- Both inputs have a 'bar' above them, indicating that they are 'active low' inputs. This means that they are activated when the input signal is logic **0**.
- Both outputs are labelled **Q** but one $\overline{Q}$ is the inverse of the other. The outputs should always be at opposite logic levels.
- When output **Q** is logic **1**, it is 'set'. When it is logic **0**, it is 'reset'.

The $\overline{S}$-$\overline{R}$ flip-flop can be implemented using just two NAND gates:



The truth table for the NAND gate is given below:

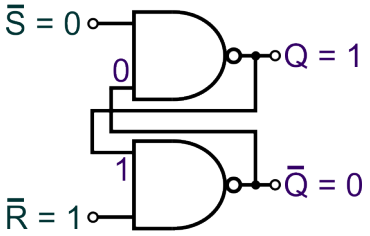
Inputs		Output
B	A	Q
0	0	1
0	1	1
1	0	1
1	1	0

One way to view the behaviour of the NAND gate is as follows:

Its output is logic **1** when either (or both) input(s) are logic **0**.

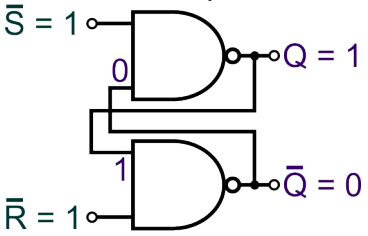
The behaviour of the flip-flop is best analysed by considering a sequence of input states, such as:

1. $\overline{S}$ **= 0**, $\overline{R}$ **= 1**



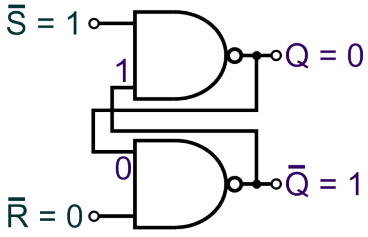
- As $\overline{S}$ **= 0**, the output (**Q**) of the upper NAND gate must be logic **1**.
- This feeds back to the input of the lower NAND gate.
  As a result, both its inputs are logic **1** giving it an output ($\overline{Q}$) of logic **0**.
- This feeds back to the input of the upper NAND gate. As a result, both its inputs are logic **0** giving it an output of logic **1**.
  Result: **Q = 1**, $\overline{Q}$ **= 0**

2. $\overline{S}$ **= 1,** $\overline{R}$ **= 1**



- From the previous conditions, **Q = 1** and $\overline{Q}$ **= 0** and these signals are fed back to the inputs of the NAND gates, as before.
- The upper NAND gate therefore has inputs **1** and **0**, giving it an output (**Q**) of logic **1** (still).
- The lower NAND gate has inputs **1** and **1**, keeping its output ($\overline{Q}$) at logic **0** (still).
  Result – no change: **Q = 1,** $\overline{Q}$ **= 0**

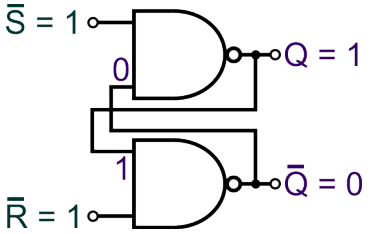
3. $\overline{S}$ **= 1**, $\overline{R}$ **= 0**



- As $\overline{R}$ **= 0**, the output ($\overline{Q}$) of the lower NAND gate must be logic **1**.
- This feeds back to the input of the upper NAND gate.
  As a result, both its inputs are logic **1** giving it an output (**Q**) of logic **0**.
- This feeds back to the input of the lower NAND gate.
  As a result, both its inputs are logic **0** giving it an output of logic **1**.
  Result: **Q = 0**, $\overline{Q}$ **= 1**

4. $\overline{S} = 1, \overline{R} = 1$

- From the previous conditions, **Q** = **0** and $\overline{Q}$ = **1** and these signals are fed back to the inputs of the NAND gates, as before.
- The upper NAND gate therefore has inputs **1** and **1**, giving
- it an output (**Q**) of logic **0** (still).
- The lower NAND gate has inputs **1** and **0**, keeping its output ($\overline{Q}$) at logic **1** (still).

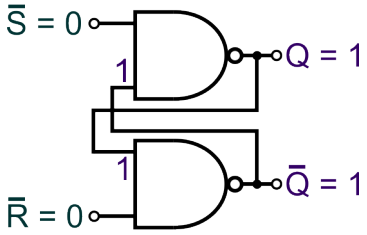


Result – no change: **Q** = **0**, $\overline{Q}$ = **1**

5. $\overline{S} = 0, \overline{R} = 0$

- As $\overline{S}$ = **0**, the output (**Q**) of the upper NAND gate must be logic **1**.
- As $\overline{R}$ = **0**, the output ($\overline{Q}$) of the lower NAND gate must also be logic **1**.

Result: **Q** = **1**, $\overline{Q}$ = **1**



We can record this sequence in a form of truth table:

State	Inputs		Outputs	
	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
1	0	1	1	0
2	1	1	1	0
3	1	0	0	1
4	1	1	0	1
5	0	0	1	1

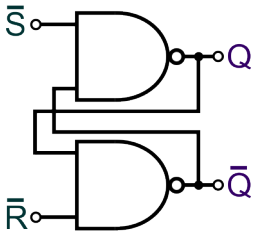
Notes:

- State 1 – logic **0** at the $\overline{S}$ input sets the output. In other words, the input is 'active-low'.
- States 2 and 4 – When both inputs are logic **1**, the outputs are unchanged from the previous state.
  This is known as the latching combination.
- State 3 – the reverse of state 1 – logic **0** at the $\overline{R}$ input resets the output.
- State 5 – **This is the problem state.**
  We specified earlier that Q and $\overline{Q}$ are in opposite states.
  The electronics doesn't object to this situation. Nothing short-circuits. Nothing overheats. The problem is in the way we defined the behaviour of the system. It means that we cannot allow this combination of inputs.

We have created a simple latch. It preserves a previous event until a reset signal is applied.
This is useful in many applications, such as a burglar alarm. A switch contained in the door frame could trigger this latch when the door opens. The alarm continues to sound even when the door is closed again. The reset switch could be located some distance from the door.

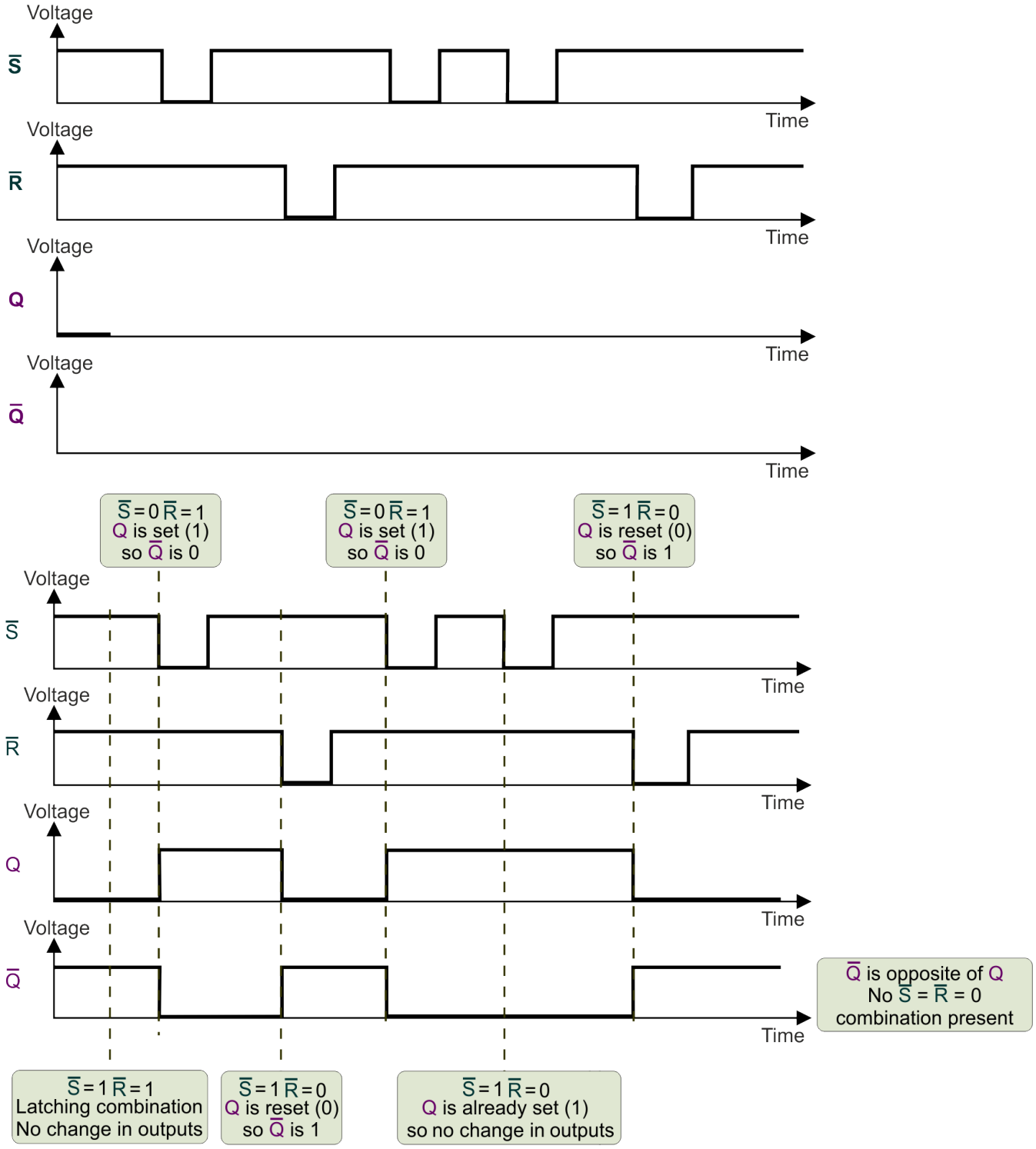
The behaviour of this system can also be shown using timing diagrams, as the following sample question shows.

Example 1:

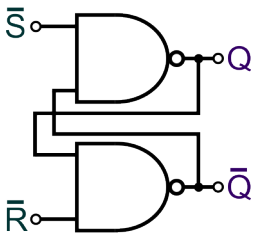


The graphs show the logic levels applied to the inputs of the flip-flop.
The **Q** output is initially at logic **0**.

Complete the graphs to show the resulting output signals at **Q** and $\overline{Q}$.



Exercise 3.1

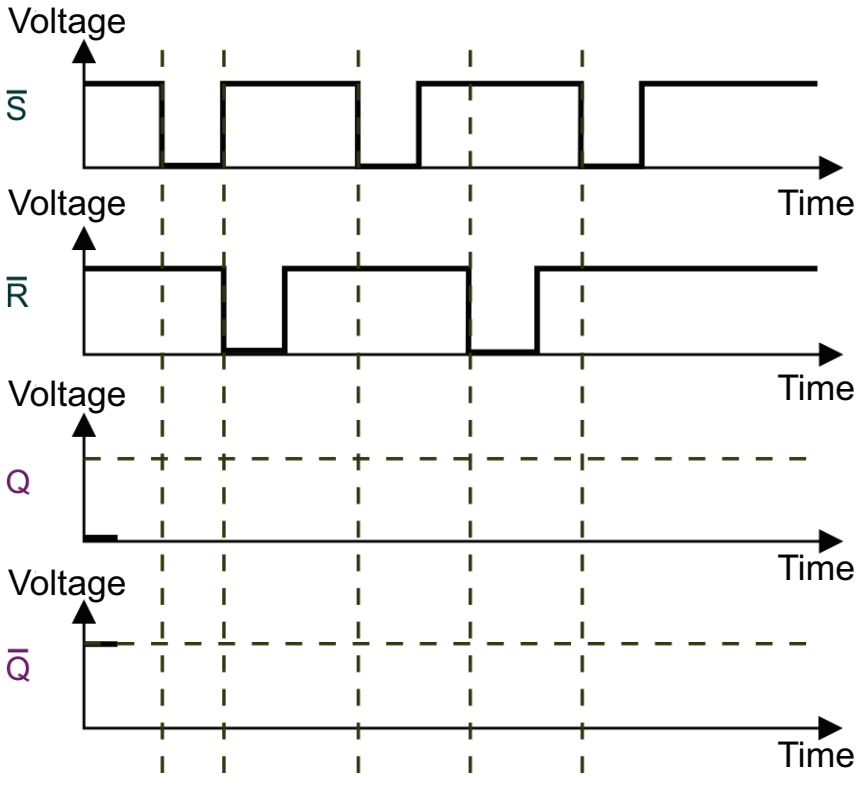


The graphs show two input sequences applied to the inputs of the flip-flop. In both cases, the **Q** output is initially at logic **0**.

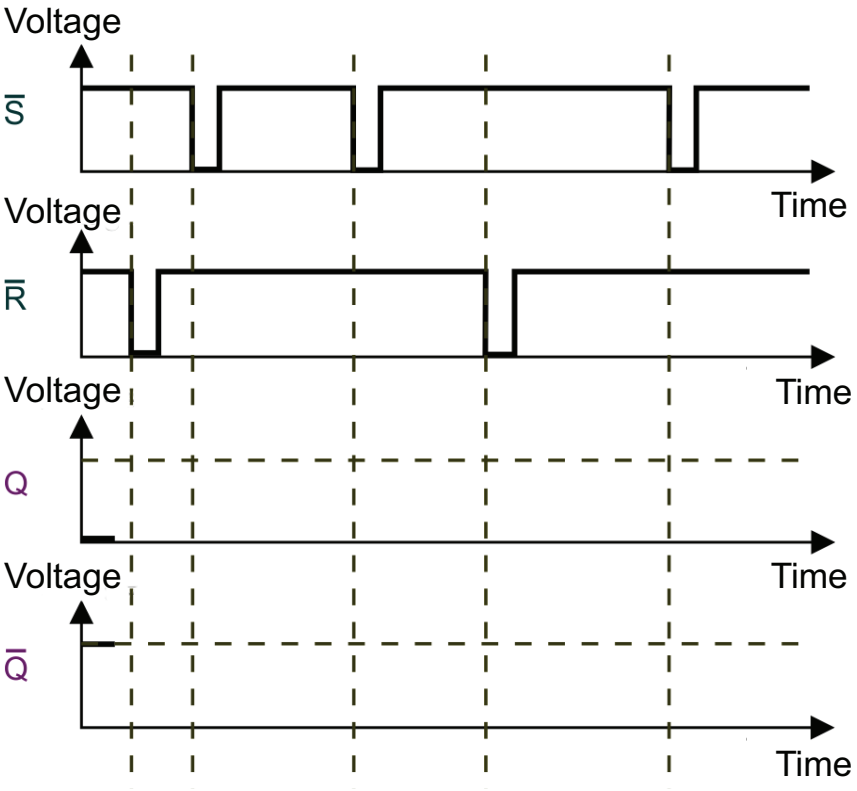
Complete the graphs to show the resulting output signals at **Q** and $\overline{Q}$.

The dotted lines are there to assist.

a)



b)

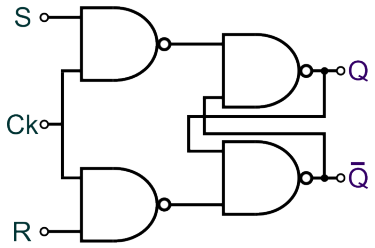


Limitations of the Simple Latch

1. The **Q** and $\overline{\mathbf{Q}}$ outputs both sit at logic **1** when both inputs are at logic **0**.
2. The inputs are active-low.
   (It is often easier to design a system based on sensors that output logic 1 activated.)
3. Changes in the outputs occur immediately in response to changes in the inputs.
   (It is often better to have control over the precise time that the output changes. For example, in computers, data exchange between sub-systems is synchronised by the central processing unit.)

Clock Input

Limitation **3** is overcome using an additional input, known as a clock input.
The diagram opposite illustrates its use with a simple latch.



The result, called the clocked **S**-**R** flip-flop, also solves limitation **2**.

The two NAND gates on the right-hand side form the simple $\overline{\mathbf{S}}$-$\overline{\mathbf{R}}$ flip-flop and work in the way discussed earlier, i.e. they are triggered by an input of logic **0**.

The NAND gates on the left send them a logic **0** signal only when both of their inputs are logic **1**. In other words, the $\overline{\mathbf{Q}}$ and **Q** outputs change only when:

- the **clock** input is logic 1 (overcoming limitation 3);
- input **S** (or **R**) is logic 1 (overcoming limitation 2).

2. Propagation Delay

Learning Objectives:

At the end of this topic you should be able to:

- draw a timing diagram to illustrate how a transition gate can produce edge triggering;
- design a transition gate to a given specification.

Edge Triggering

Use of a clock input improves the performance of the latch by ensuring that data transfer happens when a clock pulse is present. This is known as 'level-triggering' – the outputs can change whenever the clock input is at a specific logic level.

However, that clock pulse may last quite a long time and, for all of that time, the latch output is susceptible to changes in its inputs. Using 'edge-triggering', where the output can change only when the clock input is *changing* (i.e. from logic **0** to logic **1** – a rising edge – or from logic **1** to logic **0** – a falling edge– are better). One way of achieving this is to use a *transition gate*.

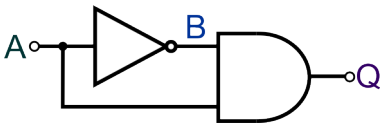
The Transition Gate

In logic circuits so far, we have assumed that changes in the output occur at the same instant as changes in the input. However this does **not** happen in reality. There is a very small delay in between these changes, called the 'propagation delay'. Typically, it is between 5ns and 10ns (1ns = 10^{-9} s).

Normally these delays are undesirable and affect the performance of a system. In logic circuits, for example, when one signal has to pass through many gates and another has to pass through only one or two, propagation delays mean that they get out of step with each other and can produce unexpected results.

A sub-system called a 'transition gate' relies on there being a propagation delay, however.

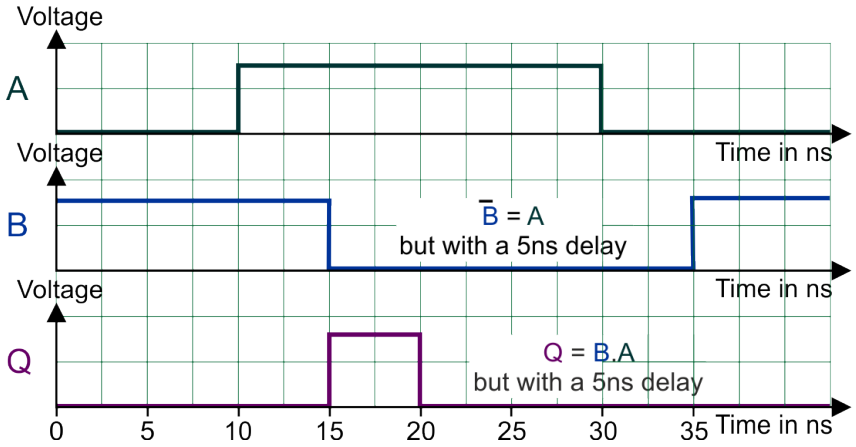
Here is an example:



At first glance, it appears that the output will always be logic **0**, since it is $A.\overline{A}$.

However, assuming a propagation delay of 5 ns for each gate, the timing diagrams below show a slightly different story:

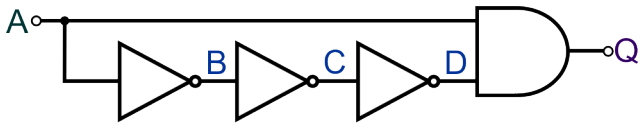
- assume that input **A** has been at logic **0** for a long time and then changes to logic **1** at time 10 ns;
- the signal at **B** changes from logic **1** to logic **0**, five nanoseconds later, i.e. at 15 ns;
- between 10 ns and 15 ns, both inputs of the AND gate sit at logic **1**, so 5 ns later, i.e. at 15 ns, the output **Q** changes to logic **1**, and at 20 ns it drops back to logic **0**;
- when input **A** returns to logic **0**, no pulse is generated because one input to the AND gate is always at logic **0**.



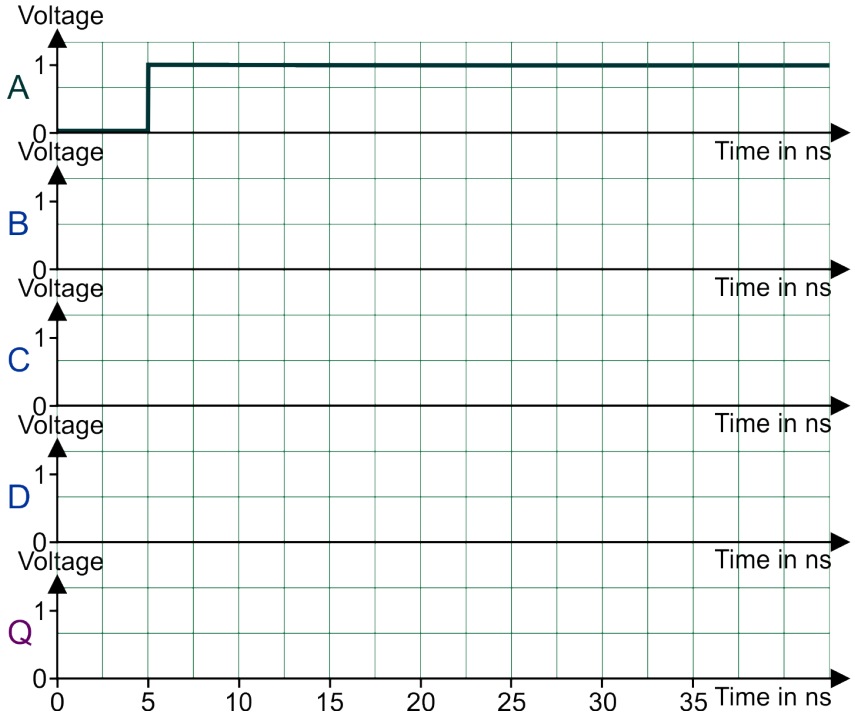
The result is that the transition gate generates a very short pulse of just 5 ns duration on the rising edge of the pulse on input **A** – the edge triggering we were looking for!

Exercise 3.2

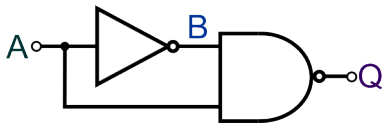
1. A transition gate is constructed from three NOT gates and an AND gate.
The propagation delay for each gate is 5 ns.



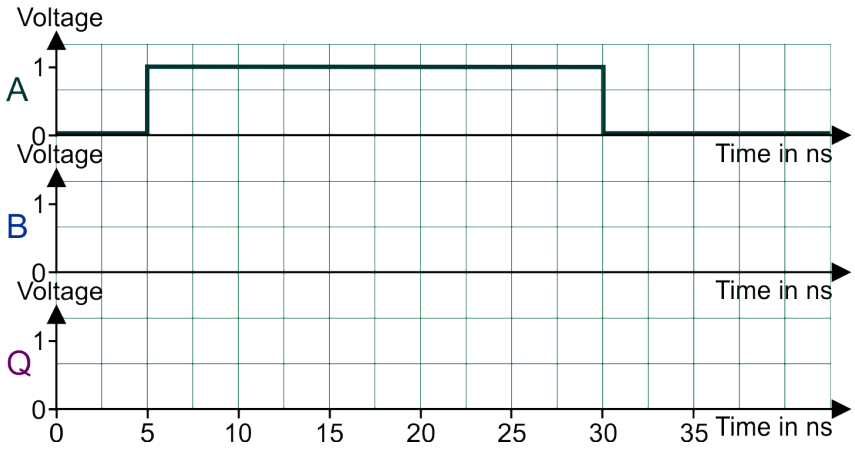
Complete the timing diagram below to show what happens after the signal at input **A** changes from logic **0** to logic **1**, as shown on the top graph.



2. A transition gate is constructed from a NOT gate and a NAND gate.
The propagation delay for each gate is 10 ns.



Complete the timing diagram to show what happens after the signal at input **A** changes from logic **0** to logic **1**, as shown on the top graph.



3. Design a transition gate to produce a logic level **0** pulse of duration 15 ns, using NAND gates only. Each NAND gate has a propagation delay of 5 ns.

3. D-Type Flip-Flops

Learning Objectives:

At the end of this topic you should be able to:

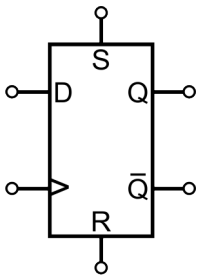
- identify and state the function of the following terminals on a D-type flip-flop:
  - clock;
  - data;
  - set;
  - reset;
  - Q output;
  - $\overline{Q}$ output.
- distinguish between the operation of the clocked data input and the set/reset inputs on a D-type flip-flop;
- complete timing diagrams to show the outputs resulting from signals applied to the clock, data, set and reset inputs.

Introduction

The limitations of the simple $\overline{S}$-$\overline{R}$ latch, no control over when switching occurs, active-low inputs and the undesirable state where the outputs are both logic **1**, are overcome in a more advanced device called the D-type flip-flop, or D-type latch.

It has four input terminals and two output terminals. The pinout is shown below.

Key:



- '**D**' – the 'data' **input**. Incoming data is ether logic **0** or logic **1**.
- '**>**' – the 'clock' **input**. This determines when data transfer takes place.
- '**S**' – the 'set' **input**. This forces the Q output to logic **1** and operates independent of the clock signal.
- '**R**' – the 'reset' **input**. This forces the Q output to logic **0** and operates independent of the clock signal. (Some devices have an 'active-low' reset. These reset when the reset input receives a logic 0 signal. This behaviour is shown by the presence of a bar – i.e. '$\overline{R}$'.
- '**Q**' – the principal **output**.
- '$\overline{Q}$' – the second **output**. This always has the opposite logic level to the Q output.

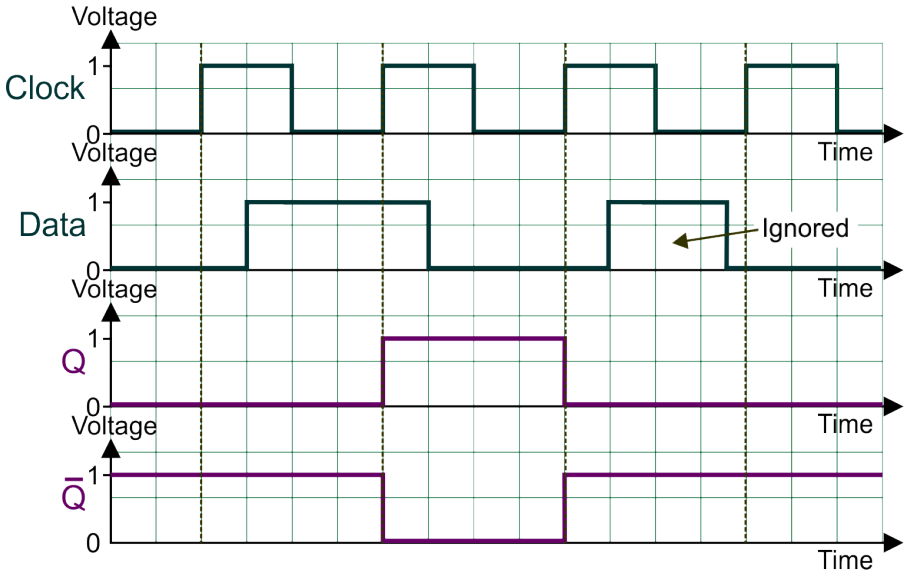
This course assumes that all D-type flip-flops are rising-edge triggered (also called 'positive-edge triggered'). This behaviour is explored in the following section.

D-type Flip-flop Behaviour

1. The clock input controls the data transfer:

On the rising edge of the clock pulse signal, the logic level present on the data input is transferred to the Q output.

The next timing diagram illustrates this action:



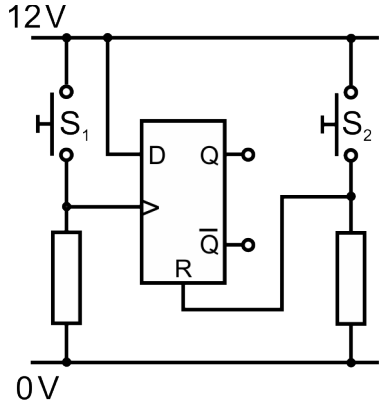
Features:

- The second data pulse has no effect on the output as it occurs between clock-pulse rising edges.
- The $\overline{\mathbf{Q}}$ output is always the opposite of the **Q** output.

This graph illustrates using a D-type flip-flop for data transfer. This is used in devices such as shift-registers and simple electronic memory. Data is moved into the device and stored there until another clock pulse is received.

Another common application is the latch. Here the 'data' input is held at logic **1**, usually by connecting it to the positive supply rail. When the D-type flip-flop receives the first clock pulse, the output 'latches' at logic **1**. Further clock pulses have no effect. Only activating the reset changes the output.

The next diagram shows the circuit diagram for a latch based on a D-type flip-flop:

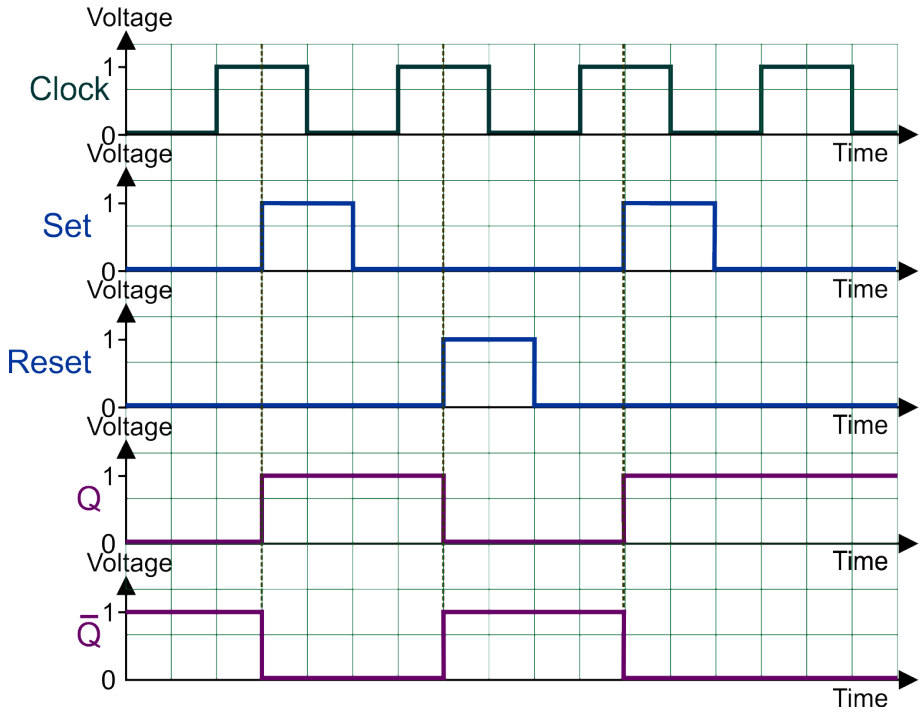


Pressing switch $\mathbf{S_1}$ sets the latch (**Q** = logic **1**).
Pressing switch $\mathbf{S_2}$ resets the latch (**Q** = logic **0**).

2. The set and reset inputs ignore the clock:

If the reset is 'active-high':
a logic 1 signal on the set input makes the Q output go to logic 1 immediately;
a logic 1 signal on the reset input makes the Q output go to logic 0 immediately.
Otherwise, if the reset is 'active-low, a logic 0 signal makes the Q output reset.

The next timing diagram illustrates this for an 'active-high' device:

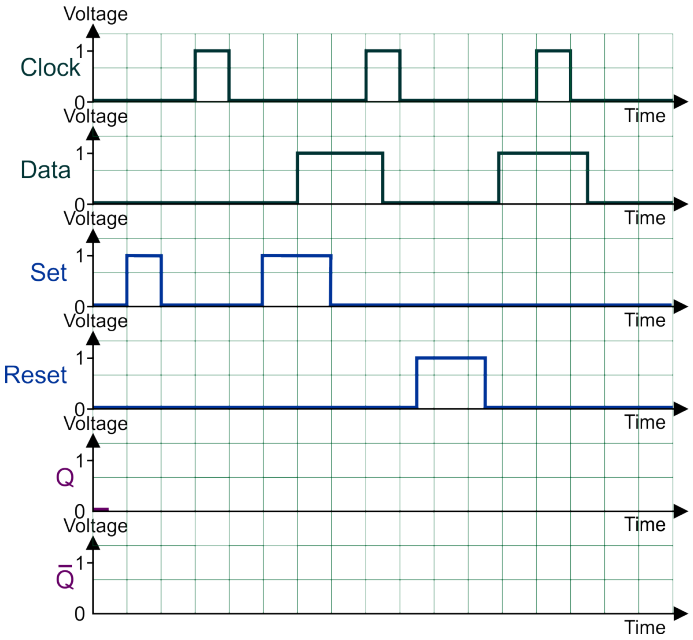


Example 1:

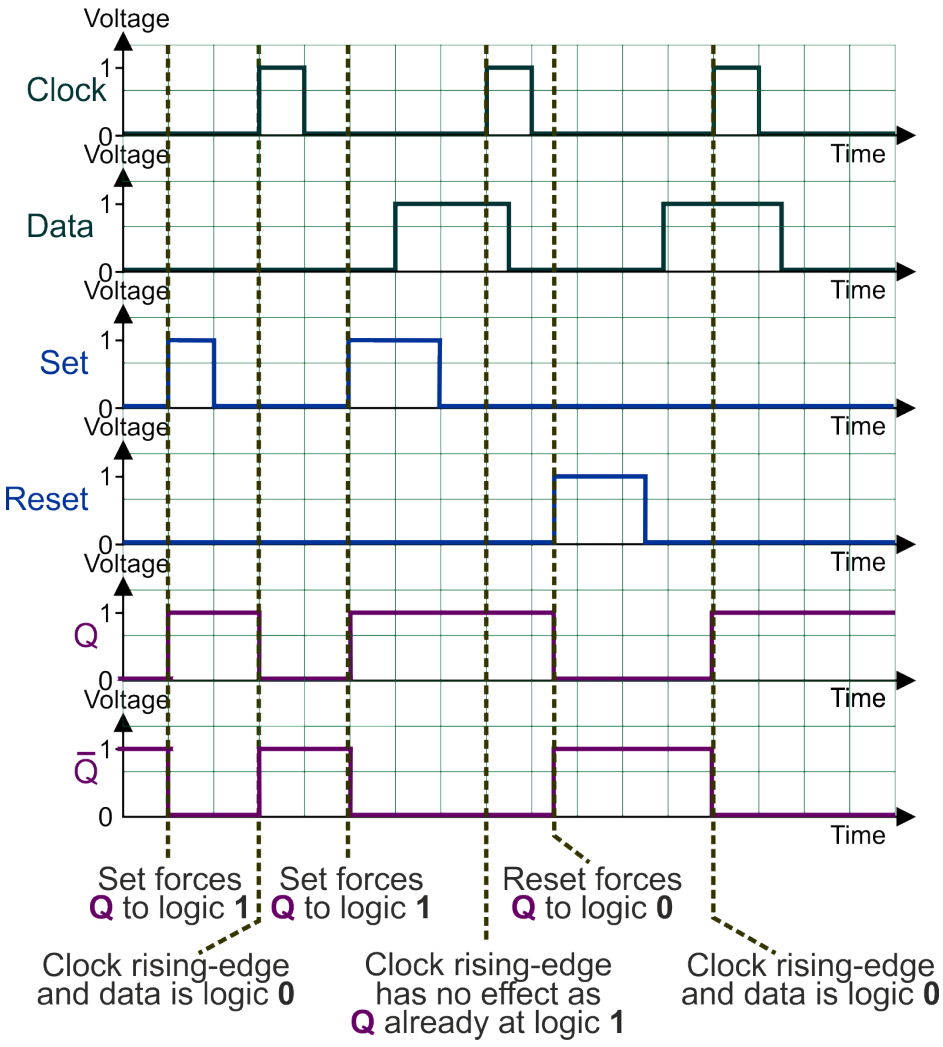
The graphs show the signals applied to the clock, data, set and reset inputs of a rising-edge triggered D-type flip-flop.

Complete the timing diagram to show the resulting effect on the **Q** and $\overline{\mathbf{Q}}$ outputs.

The **Q** output is initially at logic **0**.

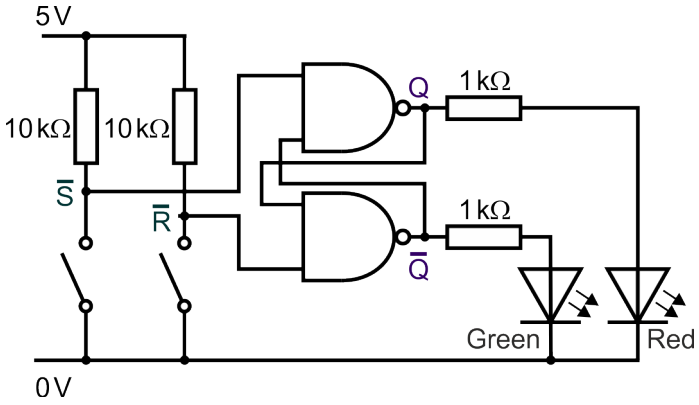


The solution is obtained by considering the combined effects of the clock, data, set and reset signals.



Investigation 3.1

1. Set up the $\overline{S}$-$\overline{R}$ flip-flop shown opposite:



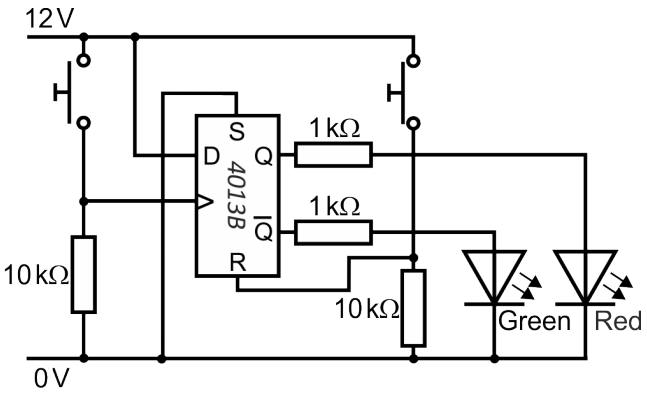
Complete the table for the input sequence provided.

Remember:

- switch **open** – input is logic **1**;
- LED **on** – output is logic **1**.

State	Inputs		Outputs	
	$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
1	0	1		
2	1	1		
3	1	0		
4	1	1		
5	0	0		

2. Set up the D-type latch shown opposite:



Complete the table for the input sequence provided.

Remember:

- switch **open** – input is logic **0**;
- LED **on** – output is logic **1**.

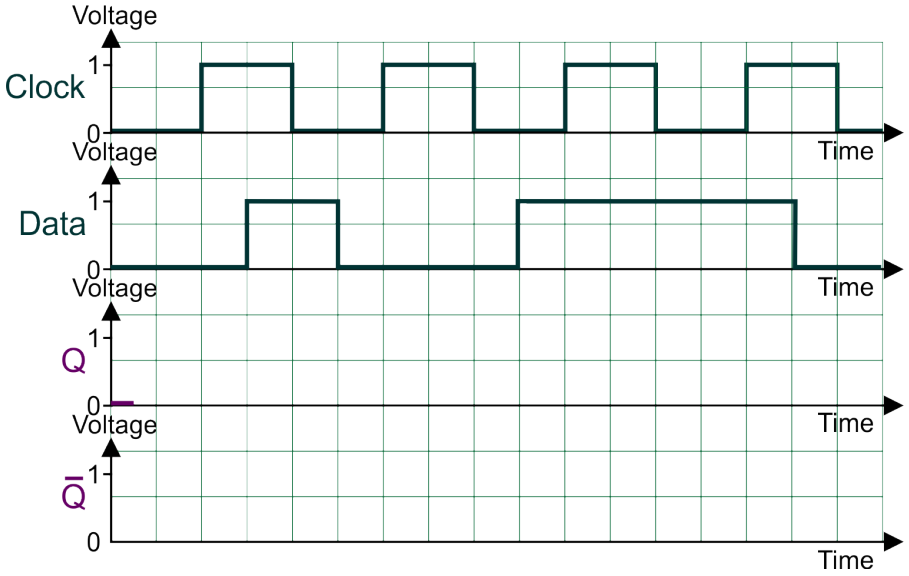
State	Inputs		Outputs	
	Clock	R	Q	$\overline{Q}$
1	0	0		
2	1	0		
3	0	0		
4	0	1		
5	0	0		

Exercise 3.3

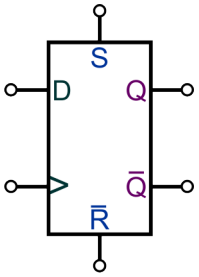
1. The graphs show the signals applied to the clock and data inputs of a rising-edge triggered D-type flip-flop. The set and reset inputs are both active-high and, held at logic 0, are inactive.

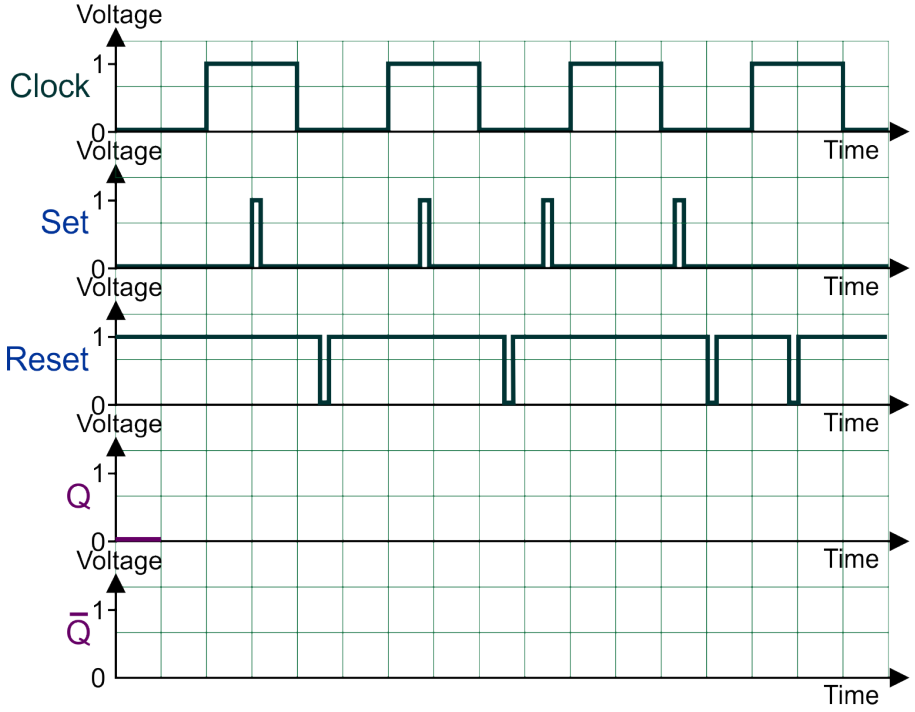
Complete the timing diagrams to show the resulting effect on the **Q** and $\overline{\mathbf{Q}}$ outputs.

The **Q** output is initially at logic **0**.



2. The graphs show the signals applied to the clock, and the set and reset inputs of a rising-edge triggered D-type flip-flop, shown opposite. The data input is held at logic **0** throughout.
Complete the timing diagrams to show the resulting effect on the two outputs.
(Hint – notice the bar over the reset pin!)
The **Q** output is initially at logic **0**.



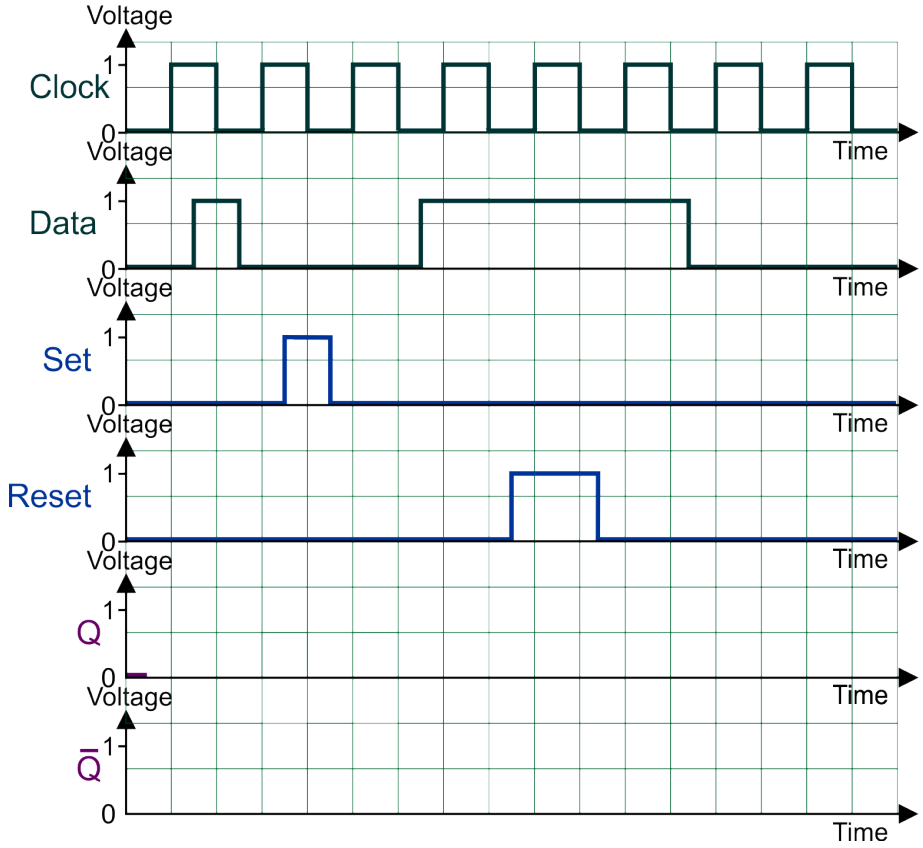


3. The graphs show the signals applied to the clock and data inputs of a rising-edge triggered D-type flip-flop.

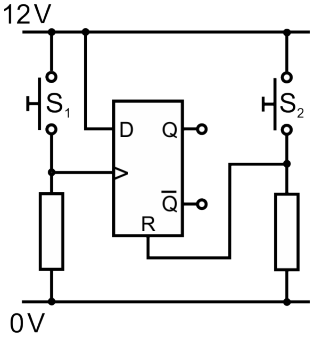
The set and reset inputs are both active-high and, held at logic 0, are inactive.

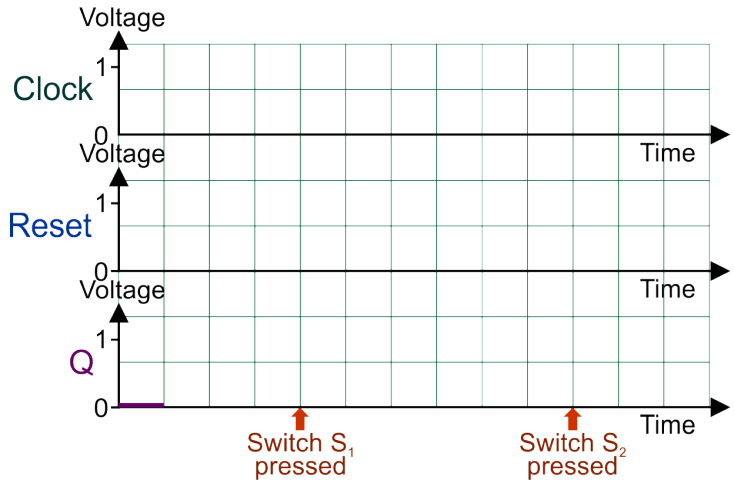
Complete the timing diagrams to show the resulting effect on the **Q** and $\overline{\mathbf{Q}}$ outputs.

The **Q** output is initially at logic **0**.



4. The diagram shows the circuit diagram for a latch based on a D-type flip-flop.
Complete the timing diagrams to show the effect of pressing switches $\mathbf{S_1}$ and $\mathbf{S_2}$. The **Q** is initially at logic **0**.





4. Number Systems

Learning Objectives:

At the end of this topic you should be able to:

- convert between binary, decimal, and hexadecimal number systems.

Introduction

Electronic counting systems are widely used. In industry, for example, they count items passing down a conveyor belt into boxes and send a signal control system when a box is full. Speed and accuracy are important and electronic counters are far faster and more accurate than humans.

Number Systems

Decimal system:

From an early age, we learn to count in the decimal system using the ten digits, '0' to '9'.
We call these 'units'.
When we reach '9', we start again in the 'units' column, with '0' and add one to the 'tens' column (which previously contained '0' though we never mentioned it!) to give '10'.
On reaching '99', we start again in the 'units' and 'tens' columns and add one to the 'hundreds' column – to give '100' – and so on.
Why 'units', 'tens' and 'hundreds'? They all relate to the 'base' of the number system, ten. (It is the decimal system after all.) It is because:

$10^0 = 1$ ('units')
$10^1 = 10$ ('tens')
$10^2 = 100$ ('hundreds')
etc.

Binary system:

This is much simpler, well at least for electronic systems. For humans, it is difficult as we quickly run into enormous numbers of digits.
There are only two digits, '0' and '1', (hence **bi**nary.) Counting in binary, we start at '0', then '1'. Now we have run out of digits and so start again in the 'units' column, with '0' and add one to the next column (which is actually the 'twos' column, **not** 'tens') to give '10', which is **NOT** ten! When we reach '11', we start with '0' in both of these columns and add one to the next column (the 'fours' column) to give '100' (which is **NOT** one hundred!)
Why 'units', 'twos' and 'fours'? The 'base' of the number system is two and

$2^0 = 1$ ('units')
$2^1 = 2$ ('twos')
$2^2 = 4$ ('fours')
etc.

Hexadecimal system:

This is useful as it allows humans to collapse the enormous number of binary digits into a manageable few. There are sixteen digits ('hexadecimal' means sixteen.) Boringly, we use the same digits as far as '9' and then use the letters 'A' (for ten), 'B' (for eleven), 'C' (for twelve), 'D' (for thirteen), 'E' (for fourteen) and 'F' (for fifteen).
The same rules apply. When we get to 'F', we reset the units to '0' and add one to the next column, the 'sixteens' column. On reaching 'FF', we reset both the 'units' and 'sixteens' and add one to the next column, the 'two-hundred-and-fifty-sixes' column.
Why 'units', 'sixteens' and 'two-hundred-and-fifty-sixes'?
The 'base' of the number system is sixteen and

$16^0 = 1$ ('units')
$16^1 = 16$ ('sixteens')
$16^2 = 256$ ('two-hundred-and-fifty-sixes')
etc.

How do we know what number system we are using? Where there is room for error, we add a subscript to the number– – '$_2$' for binary,'$_{10}$' for decimal and '$_{16}$' for hexadecimal (although there are other notations.)
There are many other number systems, but these three are of greatest use in electronics. Digital electronic systems rely exclusively on the binary system. Basically, an electronic circuit can be 'off' or 'on'. We can represent the binary numbers '0' and '1' by 'off' and 'on'. Binary numbers quickly become unmanageable for humans. We can cope with visualising '52_{10}' weeks in a year, but struggle with the binary equivalent – '110100_2' weeks in a year. Hexadecimal reduces the number of digits hugely – there are '34_{16}' weeks in a year.
The following table compares these three number systems:

Decimal	Binary	Hexadecimal
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10
17	10001	11
18	10010	12
19	10011	13
20	10100	14
21	10101	15

Features:

- The least significant bit ('units' column) in binary numbering alternates '0' / '1' / '0' / '1' etc.
- The next column on the right in binary numbering alternates every other time – '0' '0' / '1' 1' and so on.
- One hexadecimal digit replaces up to four binary digits – e.g. 'F_{16}' replaces '1111_2'.

Place Values

Number system vocabulary uses the term 'place value' to show that, for example, '1' in the 'hundreds' column is worth more than '1' in the 'units' column.

In the **decimal** system, the place values for the first four places (in decreasing order) are:
'thousands' / 'hundreds' / 'tens' / 'units'

As the following table shows, the number '1011_{10}' means ' one thousand, no hundreds, one ten and one unit' – but we knew that, as we've been using the decimal system for most of our lives!

Decimal Place Values			
10^3	10^2	10^1	10^0
Thousands	Hundreds	Tens	Units
1	0	1	1

In the **binary** system, things are less familiar. The place values for the first four places are:
'eights' / 'fours' / 'twos' / 'units'
The next table shows that the number '1011_2' means 'one eight, no fours, one two and one one'.
In decimal, this is the number eleven.

Binary places values			
2^3	2^2	2^1	2^0
Eights	Fours	Twos	Units
1	0	1	1

Possibly more mysterious, the **hexadecimal** (or 'hex') system has corresponding place values:
'four-thousand-and ninety-sixes' / 'two-hundred-and-fifty-sixes' / 'sixteens' / 'units'

The next table shows that the number '1011_{16}' means 'one four-thousand-and ninety-six, no two-hundred-and-fifty-sixes, one sixteen and one one'.
In decimal, this is the number four thousand, one hundred and thirteen.

Hexadecimal places values			
16^3	16^2	16^1	16^0
Four thousand and ninety sixes	Two hundred and fifty sixes	Sixteens	Units
1	0	1	1

Converting between Number Systems

The clue is to remember the place values. The following sample calculations will make the process clear.

Example 1:

Convert the binary number 10010011_2 into the equivalent:

a) decimal number;
b) hexadecimal number.

a) Looking at place values, this binary number is:

	1	0	0	1	0	0	1	1
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal place value →	128	64	32	16	8	4	2	1

Ignoring columns in the first row that contain a zero, the decimal number is:

$$128 + 16 + 2 + 1 = 147_{10}$$

b) Method 1:

Complete the table for hexadecimal place values to give a total of 147_{10}.

	16^2	16^1	16^0
Decimal place value →	256	16	1
	0	**9**	**3**

i.e. $(9 \times 16) + (3 \times 1) = 147_{10}$

As a hexadecimal number, it is 93_{16}.

Method 2:

Remember that one hexadecimal digit replaces four binary digits.

Put the binary number into groups of four digits and use the table given earlier to convert each into the equivalent hex. number.

$$10010011 \Rightarrow (1001)\ (0011) \Rightarrow (9)(3)$$

Once again, this gives an answer of 93_{16}.

Example 2:

Convert the decimal number 181_{10} into the equivalent:

a) binary number;

b) hexadecimal number.

a) Complete the table for binary place values to give a total of 181_{10}.

	1	0	1	1	0	1	0	1
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal place value →	128	64	32	16	8	4	2	1

Again ignoring columns in the first row that contain a zero, the decimal number is:

$$128 + 32 + 16 + 4 + 1 = 181_{10}.$$

b) The simplest way is to convert the binary number just obtained into hex. by grouping it into sets of four digits and then replacing each with its hex. equivalent.

$$(1011_2)(0101_2) = B5_{16}$$

Alternatively, complete the table for hex. place values to give a total of 181_{10}.

	16^2	16^1	16^0
Decimal place value →	256	16	1
	0	**B**	**5**

i.e. $(B_{16}(= 11_{10}) \times 16) + (5 \times 1) = 181_{10}$

Once again, the hex. number is $B5_{16}$.

Example 3:

Convert the hexadecimal number FC_{16} into the equivalent:

a) binary number;

b) decimal number.

a) Converting the hex. digits FC_{16} directly into binary digits, gives:

$$FC_{16} = (1111_2)(1100_2)$$
$$= 11111100_2.$$

b) The simplest way is to convert the binary number just obtained into decimal:

	1	1	1	1	1	1	0	0
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Decimal place value →	128	64	32	16	8	4	2	1

i.e. $(1 \times 128) + (1 \times 64) + (1 \times 32) + (1 \times 16) + (1 \times 8) + (1 \times 4) = 252_{10}$

Alternatively look at place values for hexadecimal numbers:

	16^2	16^1	16^0
Decimal place value →	256	16	1
	0	F	C

As a decimal, this number is: $(F_{16}(=15_{10}) \times 16) + (C_{16}(=12_{10}) \times 1) = 252_{10}$.

As a decimal number, it is 252_{10}.

Exercise 3.4

1. Convert the following decimal numbers into binary numbers.

a) 77

.....
.....
.....

b) 103

.....
.....
.....

2. Convert the following binary numbers into decimal numbers.

a) 01011001

.....
.....
.....

b) 11010111

.....
.....
.....

3. Convert the following binary numbers into hexadecimal numbers.

a) 01011001

.....
.....
.....

b) 11010111

.....
.....
.....

5. Counters

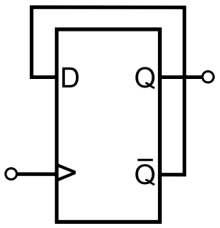
Learning Objectives:

At the end of this topic you should be able to:

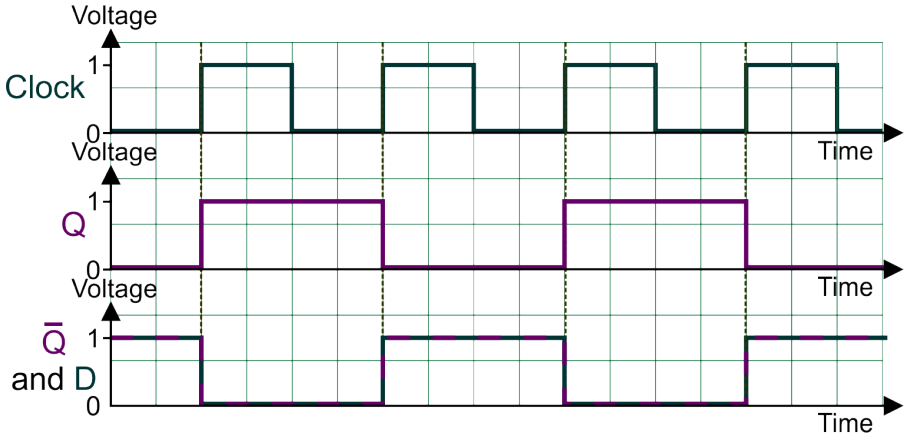
- connect a series of D-type flip-flops or counters to produce a frequency divider circuit;
- design 4-bit up and down counters based on D-type flip-flops;
- design 4-bit modulo-n counters and binary-coded decimal (BCD) counters;
- draw timing diagrams for these counters;
- describe the use of decoders and seven-segment displays;
- design systems that use a counter and combinational logic to produce a sequence of events.

The 1-bit Counter

The starting block for a counter is the edge-triggered D-type flip-flop connected as shown, with the $\overline{Q}$ output connected to the data input. (The Set and Reset inputs have been omitted as they play no part in the present behaviour. In practice, they would be connected to logic **0** and be inactive.)
This leaves one input (the clock) and one output (the **Q** output).



The timing diagram for this arrangement is shown below.

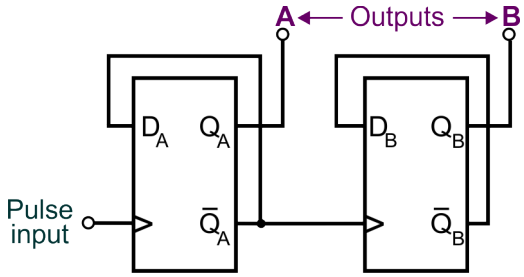


Features:

- The signal generated on the $\overline{Q}$ output is fed back to the **D** (data) input. In other words, the logic level of the **D** input is always the opposite of the **Q** output.
- Changes in the outputs happen only on rising edges of the clock pulses. When the **Q** output is logic **0**, the **D** input is logic **1**. On the rising edge of the next clock pulse, **Q** changes logic **1**, $\overline{Q}$ and **D** change to logic 0 and so on.
- As a result, the outputs 'toggle' (change state) on every clock pulse. (This sub-system is also known as a '**T-Type**' flip-flop.)
- The sub-system is also known as a '**divide-by-two**' circuit, as the output signal contains only half as many pulses as the clock input. Equally, the frequency of the output pulses is half that of the clock pulses.
- It is also known as a '**1-bit counter**' as the output alternates between '0' / '1' / '0' / '1', etc., as each clock pulse triggers it. (This was the behaviour noted earlier for the least significant bit of the binary number system.)

The 2-bit Counter

As a counter, the previous system is pointless – it can count only to 1.
Adding a second 1-bit counter improves the capabilities of the system.

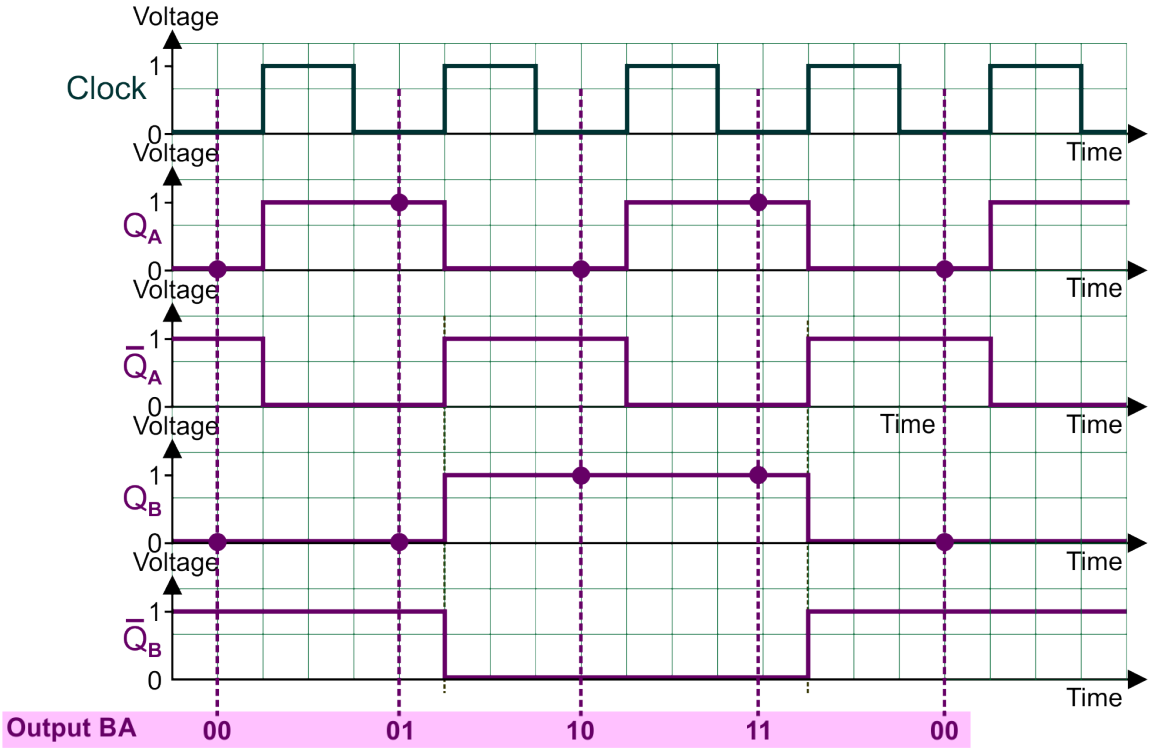


(As before, the Set and Reset inputs have been omitted.)

The '**B**' output is the most-significant bit and the '**A**' output the least-significant bit of the extended counter.

The two 1-bit counters are linked with a connection from the $\overline{Q}$ output of the first (now labelled $\overline{Q}_A$,) to the clock input of the second. In other words, the second 1-bit counter is triggered by rising edges on pulses created by the $\overline{Q}$ output of the first.

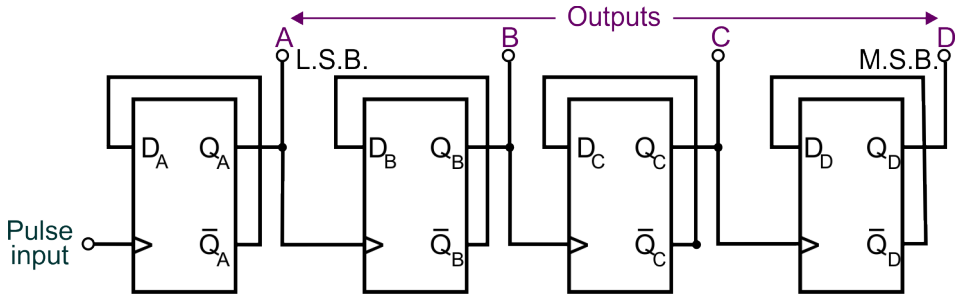
Initially, both outputs, **A** and **B**, are reset (logic **0**.) The timing diagram is shown below.



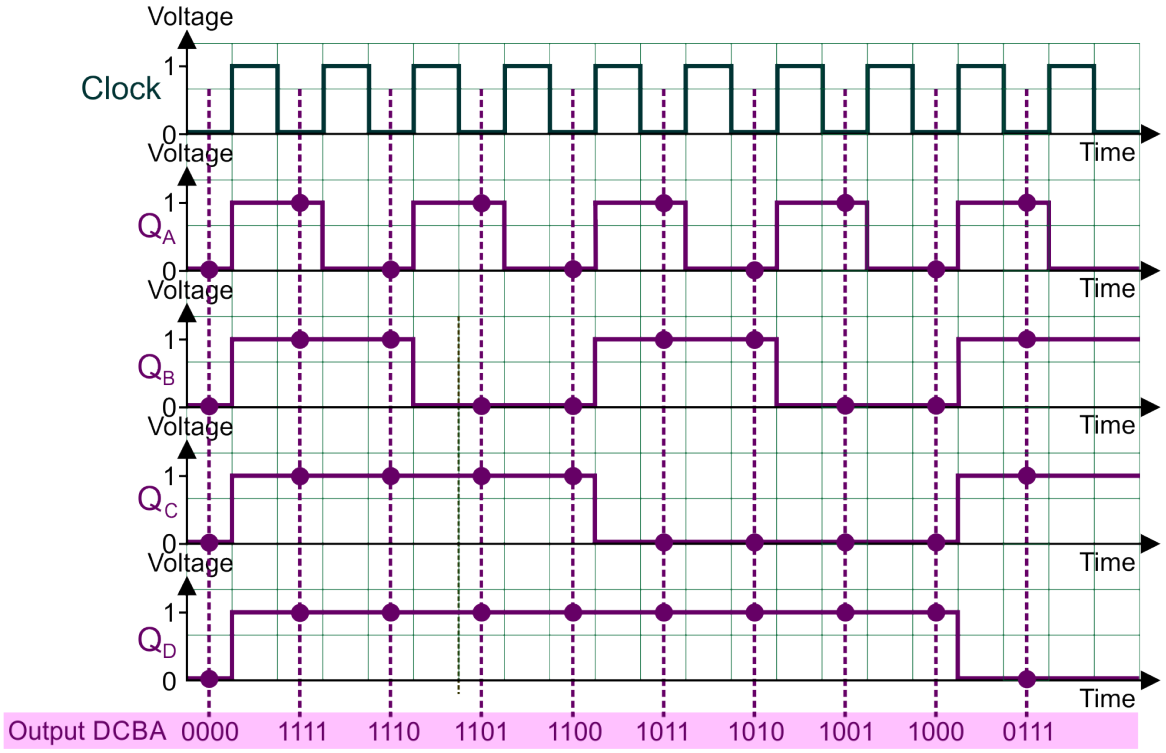
Features:

- Output Q_A creates one pulse for every two clock pulses. Put another way, the period of the Q_A pulses is twice that of the clock pulses, or the frequency of the Q_A pulses is half that of the clock pulses, (divide-by-two!)
- Output Q_B creates one pulse for every four clock pulses. The Q_B pulses have a period twice as long as the clock pulses and a frequency one quarter that of the clock pulses.
- Reading the states of Q_B and Q_A before each clock pulse gives the two-bit binary counting sequence – $00_2 \Rightarrow 01_2 \Rightarrow 10_2 \Rightarrow 11_2 \Rightarrow 00_2$ etc.
- This arrangement creates a two-bit up-counter, with Q_A as the least significant bit.

Changing the system so that each stage receives its clock pulse from the **Q** output of the previous stage, rather than the $\overline{\mathbf{Q}}$ output, produces a binary down-counter.
The diagram shows this arrangement used to make a 4-bit down-counter.



The timing diagram is shown below. It shows only the **Q** outputs. The $\overline{\mathbf{Q}}$ outputs play no part in triggering the next stage and are simply the inverse of the corresponding **Q** output.



Features:

- Once again, each stage of the counter has a period twice that of the previous stage and a frequency half of that of the previous stage.
- Reading the states of the outputs before each clock pulse gives the 4-bit binary counting sequence – $0000_2 \Rightarrow 1111_2 \Rightarrow 1110_2 \Rightarrow 1101_2 \Rightarrow 1100_2 \Rightarrow 1011_2 \Rightarrow 1010_2 \Rightarrow 1001_2 \Rightarrow 1001_2$, etc.
- This arrangement creates a 4-bit down-counter, with $\mathbf{Q_A}$ as the least significant bit (LSB) and $\mathbf{Q_D}$ as the most significant bit (MSB).

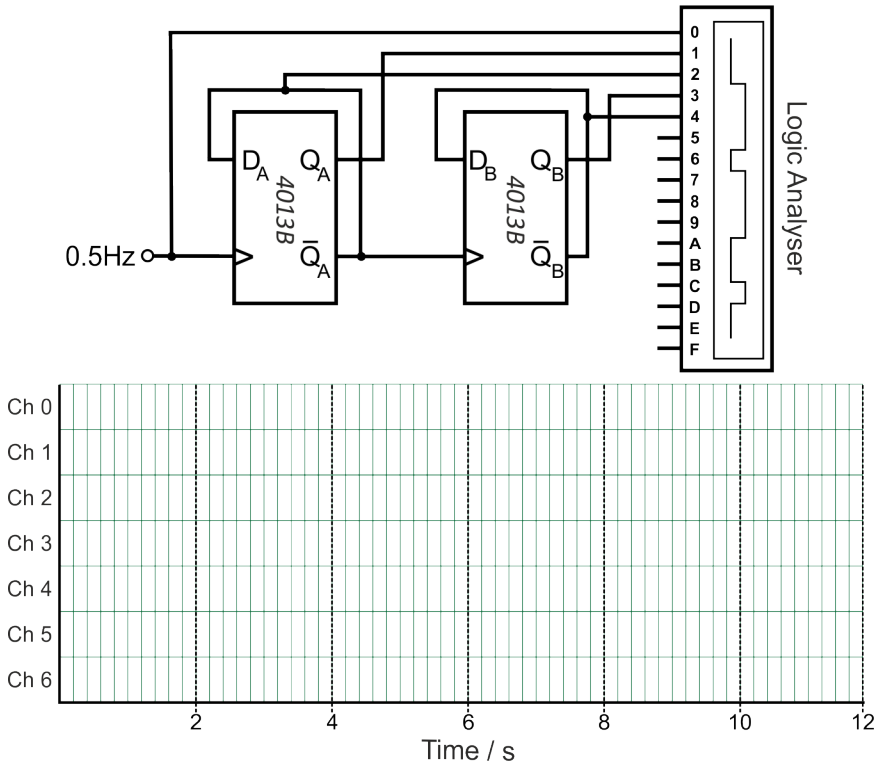
Investigation 3.2

a) Set up the 2-bit counter on your simulation software.

Make the following connections to the logic analyser:

- connect the 0.5 Hz clock to channel 0;
- connect Q_A to channel 1;
- connect $\overline{Q}_A$ to channel 2;
- connect Q_B to channel 3;
- connect $\overline{Q}_B$ to channel 4.

b) 'Right-click' on the logic analyser to select the graph and resize it to correspond to the graph grid shown below.



c) Pause the simulation after about ten seconds and copy the result onto the graph grid.

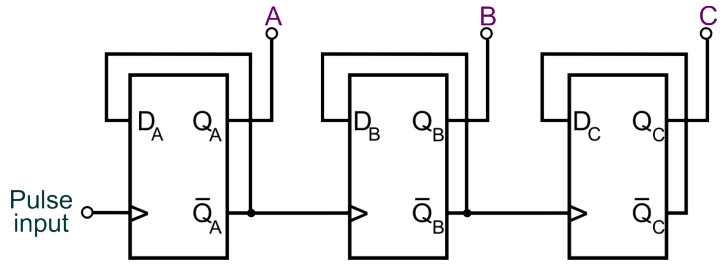
d) Compare the signals with those of the 2-bit counter analysed earlier.
Comment on your results.

.....

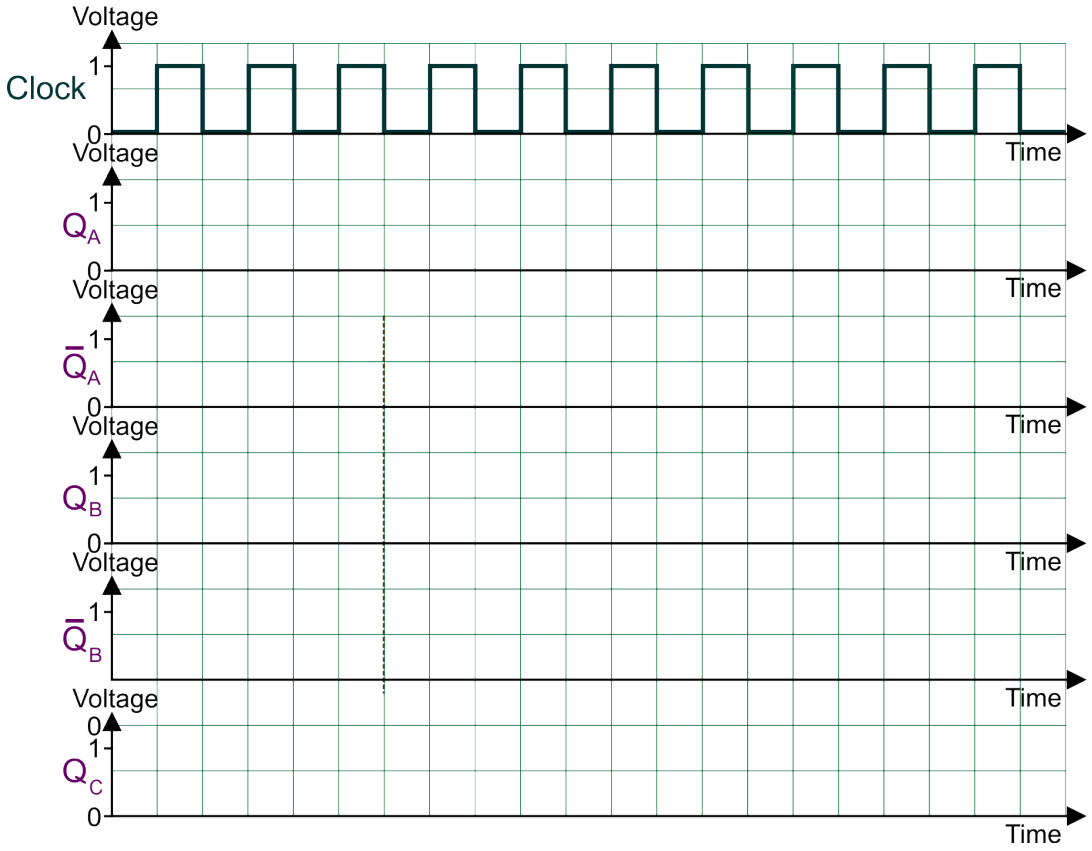
.....

Exercise 3.5

1. The diagram shows a 3-bit binary up-counter. Initially, the counter is reset.



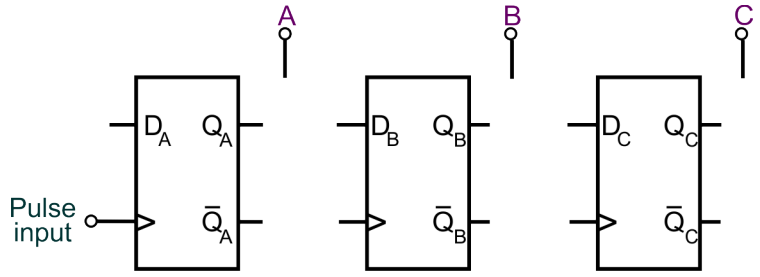
a) Complete the timing diagram for this system.



b) What is the output of the counter after the fifth clock pulse?

Output = ……..

2. Complete the diagram of a 3-bit binary down-counter.

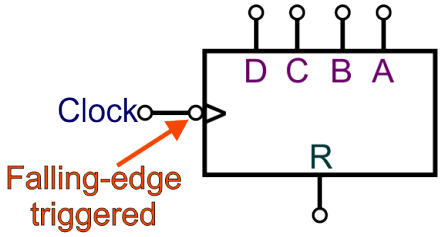


Counter ICs

There are two main types of counter, known as ripple counters and synchronous counters. Their function is identical – to count in binary – but their internal structure is different.

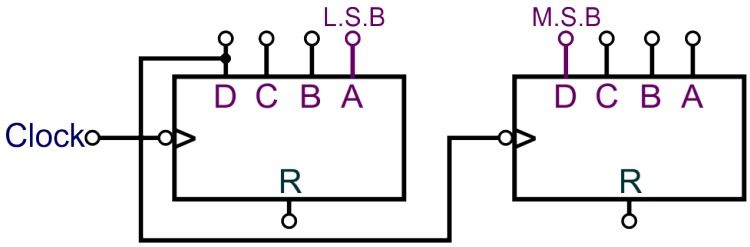
Ripple counters

A ripple counter consists of a number of 1-bit counters, linked together. The 2-bit counter just described is an example. Each stage receives its clock pulse from the output of the previous 1-bit counter. As a result, the count 'ripples' through the stages of the counter.
For simple counters, connecting 1-bit counters together in this way provides a satisfactory solution. The CMOS 4013 IC, for example, contains two D-type flip-flops. Hence, with this, it is possible to build a 2-bit counter using only one IC.
Larger counters are available where the whole circuit is contained in a single IC.
The diagram shows the symbol for a 4-bit ripple counter.



Notice the circle on the clock input, which shows that the device is falling-edge triggered, unlike the D-type flip-flop studied earlier, which is rising-edge triggered. The absence of a bar over the reset terminal, **R**, indicates that the device resets when the reset terminal receives a logic **1** signal, i.e. is 'active-high'.

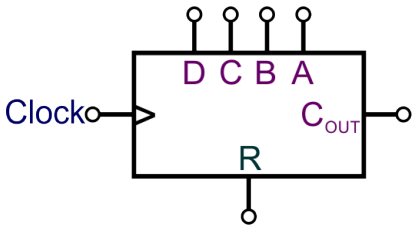
With up-counters made from individual D-types, stages are linked together by connecting the $\overline{\mathbf{Q}}$ output of one stage to the (rising-edge triggered) clock input of the next. Dedicated IC counters do not allow access to the $\overline{\mathbf{Q}}$ outputs. Instead, the clock input is made falling-edge triggered so that the signal from the **Q** output of the most significant bit of the one counter can provide correctly synchronised clock signals to the next IC. The following diagram shows how to link two 4-bit ripple counters to make an 8-bit counter. The least and most significant bits of the 8-bit counter are labelled.



Propagation delay can be a problem, however, because earlier stages of the counter must change before the clock pulses reach the later stages. The more stages (bits) there are in the counter, the bigger the problem. Eventually, with high frequency clock pulses, changes in the most significant bits caused by one clock pulse may not be completed before the outputs of the least significant bits start to change as a result of the next clock pulse.

Synchronous counters

This issue is avoided in the synchronous counters. The internal circuitry is designed so that all stages of the counter change at the same time (in synchrony!)
As a result, there is no problem when used at high frequencies.



The circuit symbol for a 4-bit counter IC is shown opposite:

This time, the counter is rising-edge triggered.
For most synchronous counters, the reset terminal is 'active-high'.
The 'carry out' terminal is used to link together synchronous counter ICs to make bigger counters.

BCD

In many applications, we wish to view the count as it takes place.
This is facilitated using a number code known as Binary-Coded Decimal (BCD).
As its name suggests, it is obtained by writing each decimal digit as its binary equivalent.

For example:

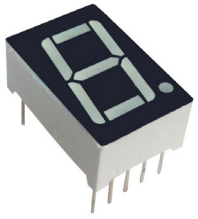
Decimal number	BCD equivalent
7	0111
24	0010 0100
85	1000 0101
326	0011 0010 0110

Method:

- To convert '24_{10}', the '2' converts to '0010_2' and the '4_{10}' to '0100_2' giving the BCD equivalent of '0010 0100'.
- To convert '326_{10}', the '3_{10}' converts to '0011_2', '2_{10}' to '0010_2' and '6_{10}' converts to '0110_2' giving the BCD equivalent of '0011 0010 0110'.

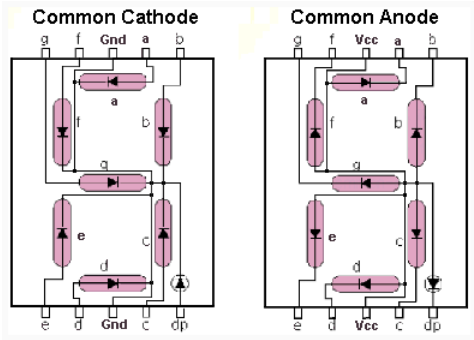
Notice that, in general, BCD requires more bits to represent a number then pure binary. For example, the binary equivalent of '326_{10}' is '101000110_2' – nine bits, not the twelve bits used by BCD.

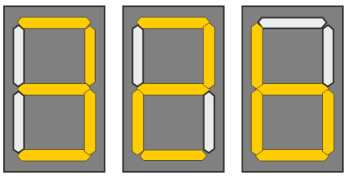
Seven-segment displays



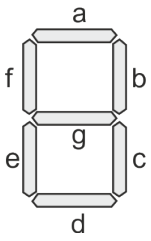
To display the count, a seven-segment display can be used. In its simplest form, it consists of seven bar-shaped LEDs (and a LED for the decimal point).
Each is controlled independently. In the 'common-cathode' version, a LED lights when it receives a logic **1** signal. In the 'common-anode' version, it lights when it receives a logic **0** signal.

The next diagram shows the internal structure of a seven-segment display – for information only.





By lighting different combinations of LEDs, the decimal number set can be displayed. Here, a set of three displays shows the number ‘326’. To do this, appropriate logic signals are applied to segments ‘a’ to ‘g’. To create the number ‘3’ on a common-cathode display, the following combination of signals is sent to the display:



Segments						
a	b	c	d	e	f	g
1	1	1	1	0	0	1

Exercise 3.6

1. Convert the following numbers to BCD:

a) 6_{10}

.....

.....

b) 29_{10}

.....

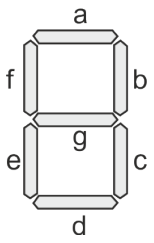
.....

c) 157_{10}

.....

.....

2. Complete the table to show the signals that must be sent to each of the following to make it display the number '5':



a) a **common-cathode** seven-segment display.

Segments						
a	**b**	**c**	**d**	**e**	**f**	**g**

b) a **common-anode** seven-segment display.

Segments						
a	**b**	**c**	**d**	**e**	**f**	**g**

Decoder-driver IC

The job of creating signals for the seven-segment display from the counter outputs is done by a device called a 'seven-segment decoder-driver' IC. There are two versions, one for 'common-cathode' and one for 'common-anode' displays. For a 'common-anode' device, the decoder must output a logic **1** signal to make a segment light. For a 'common-anode' device, it must output a logic **0** signal to make a segment light.

These devices are combinational logic systems, having four inputs (from the counter IC) and seven outputs (one for each LED in the seven-segment display).

Part of the truth table for a common-cathode device is shown opposite. (For a common anode device the decoder outputs would be reversed.)

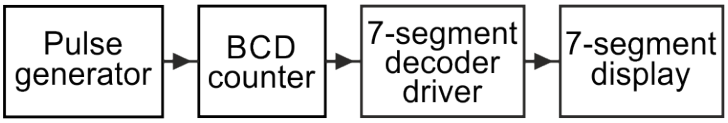
Counter Output		Decoder Output						
Decimal	Binary	a	b	c	d	e	f	g
0	0000	1	1	1	1	1	1	0
1	0001	0	1	1	0	0	0	0
2	0010	1	1	0	1	1	0	1
3	0011	1	1	1	1	0	0	1
4	0100	0	1	1	0	0	1	1

These devices perform two functions:

- they decode the BCD counter outputs into the seven input signals for the display;
- they boost the current to the displays.

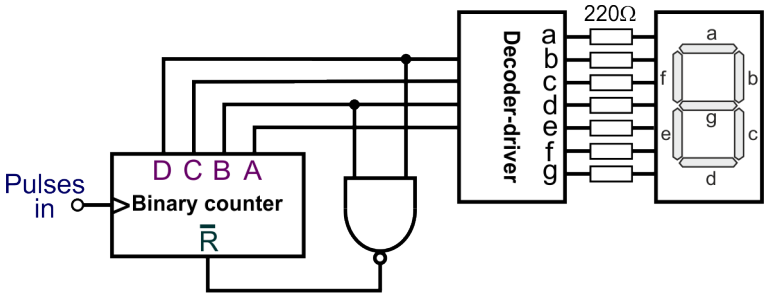
Decimal Counting System

The block diagram for a decimal counting system follows:



The pulse generator is shown as the input device though other sensing circuits can be used, providing they output digital signals.

The following circuit diagram shows how a 4-bit binary counter can be used as a BCD counter. It is connected to a decoder-driver IC and display. Current-limiting resistors are added to protect the LED segments. The resistor value depends on the power supply voltage. For a 9 V power supply, the 220 Ω resistors shown in the diagram are suitable.



Practical Counters

Counters are available with a range of output bits, e.g. four, eight, twelve and fourteen. We will only consider 4-bit counters.

In practice, counters come with a variety of functions, including:

- Count up / count down:
  The action is determined by the logic level present on a separate input pin.
- Presettable
  The output can be preset to the number present on the 'Load' inputs, 'L1' to 'L8'.
- BCD count / Binary count
  A separate input, allows the counter to be configured as either a BCD counter or as a binary counter.
- Built in decoders
  Some BCD counters have a built-in decoder for seven-segment displays, and provide the outputs ready for direct connection to the display.

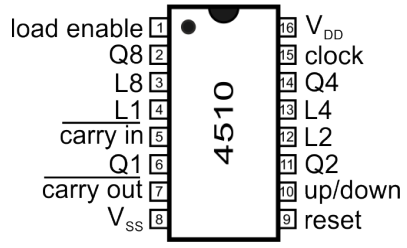
Some popular counters are:

CMOS 4510

This BCD counter offers both 'Count up / count down' and presetting:

Features:

- When the 'up/down' terminal is at logic **1**, the counter counts up.When at logic **0**, it counts down.
- When the 'load enable' pin is at logic 1, the number present on 'L8', 'L4', 'L2' and 'L1' is transferred to outputs 'Q8', 'Q4', 'Q2' and 'Q1'.
- Cascading several counters together is achieved using the 'carry in' and 'carry out' terminals.



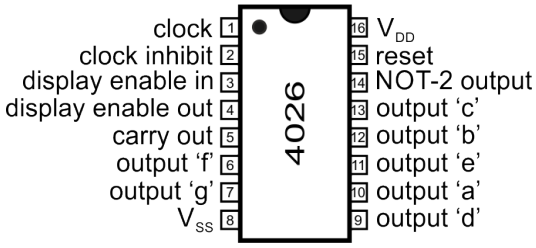
Where a binary counter, rather than BCD, is prefered, the CMOS 4516 binary counter is pin-compatible with the 4510.
The 4510 (or 4516) could be replaced with a 4029 IC which is pin-compatible except that the reset pin is replaced with a B/D input. A logic **1** input causes it to count in binary and a logic **0** to count in BCD.

CMOS 4017

This is a decade counter. It has one clock input and ten outputs. Only one output is high at a time. Each output is activated in turn on the rising edge of a clock pulse.
When the EN (enable) pin is at logic **1**, the display freezes, and when at logic **0** it allows each output to go high in turn.
(The 4017 counter will not be examined on written papers but is very useful in designs for electronic games or toys.)

CMOS 4026

This BCD counter has a built-in decoder to control common-cathode seven-segment displays.

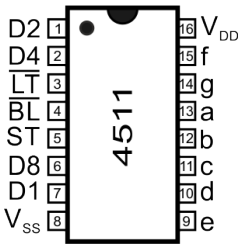


Features:

- When the 'clock inhibit' (INH) pin is at logic **1**, the count is frozen – clock signals have no effect on the counter.
- When the 'display enable in' (DEI) pin is at logic **1**, the display is enabled, meaning that the current number is displayed. This is a power saving measure – the display lights only when required.
- The 'display enable out' (DEO) terminal controls other counters linked to it.
- The 'carry out' (CO) terminal produces a pulse when the count reaches ten. This is used to enable other counters linked to it, providing clock pulses for 'tens' hundreds' etc.

Practical decoder-driver IC

The 4511 decoder-driver is a popular device for controlling common-cathode seven-segment displays.



Features:

- The BCD inputs are labelled D1, D2, D4 and D8.
- Outputs 'a' to 'g' connect to the corresponding segments of the display.
- When the 'ST' terminal is at logic **0**, the BCD input is decoded and appropriate signals are sent to the seven-segment display.
- The 'lamp test' function lights all segments of the display when the $\overline{LT}$ terminal is at logic **0**. For normal operation, this terminal is held at logic **1**.
- The 'ripple blanking' function prevents the digit '0' being displayed when the $\overline{BL}$ terminal is at logic **0**. This is useful in multi-digit displays, when a number of 4511 ICs are cascaded together. Instead of displaying '000…000', use of ripple-blanking means that most displays are blank, with just one displaying '0'.

Other devices

TTL 7447

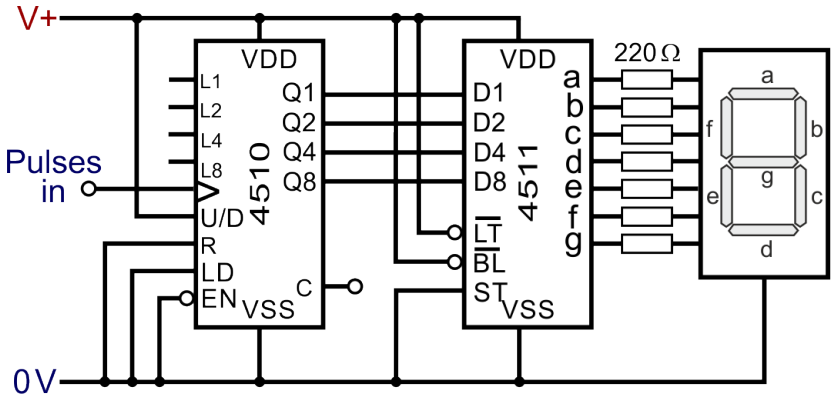
- a decoder/driver for controlling common-anode seven-segment displays.

CMOS 4543

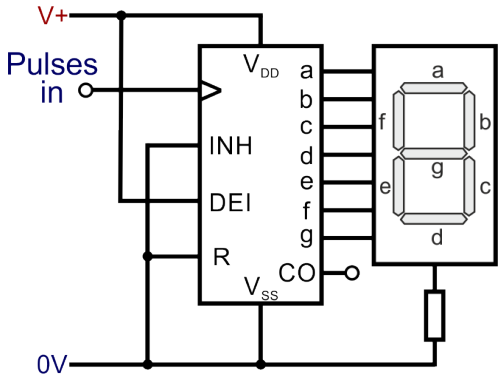
- can be used for controlling common-anode and common cathode seven-segment displays. (It has a 'PH' (phase) input, set to logic 0 for common cathode displays and to logic 1 for common anode displays.)

Practical Decimal Counting Systems

This is how a 1-bit decimal counting system is implemented using the 4510 BCD counter and 4511 decoder/driver:

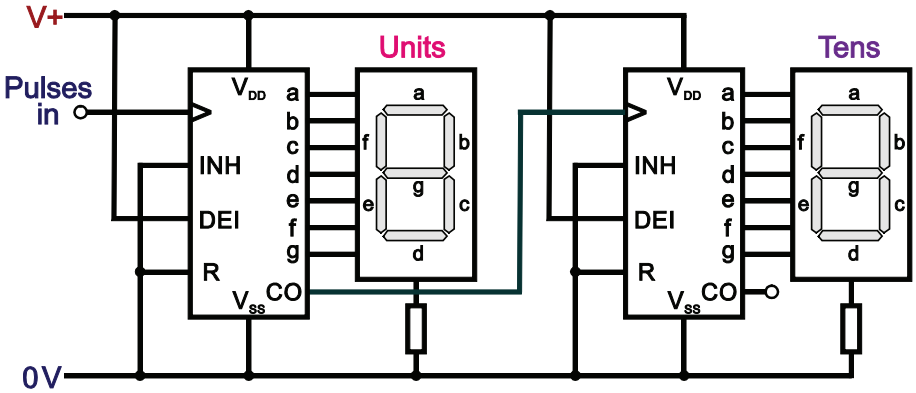


Here is a 1-bit decimal counting system using the 4026 IC:



To link together a number of 4026 counters, the 'carry out' terminal on the first is connected to the clock input on the next.

The following diagram shows how this arrangement (the green connection) is used to create a two-digit decimal display.



Note: The first of the circuits above has individual current-limiting resistors for each LED segment. In the bottom two circuits, the segments share a common current-limiting resistor. The disadvantage of this arrangement is that the current through each segment depends on how many segments are lit, causing slightly different levels of illumination for different numbers.

Modulo-n Counters

A modulo-n counter is one that has n states. For example, a 4-bit binary counter has sixteen states (from 0000_2 to 1111_2) and so is a modulo-16 counter.

Sometimes we need to produce a counter that counts up to only five or six for example. We can use a 4-bit binary counter and add a logic system that resets it at the appropriate point in the counting sequence. The aim is to reset the counter on the binary number that is one higher than the last number needed.

Sample Design

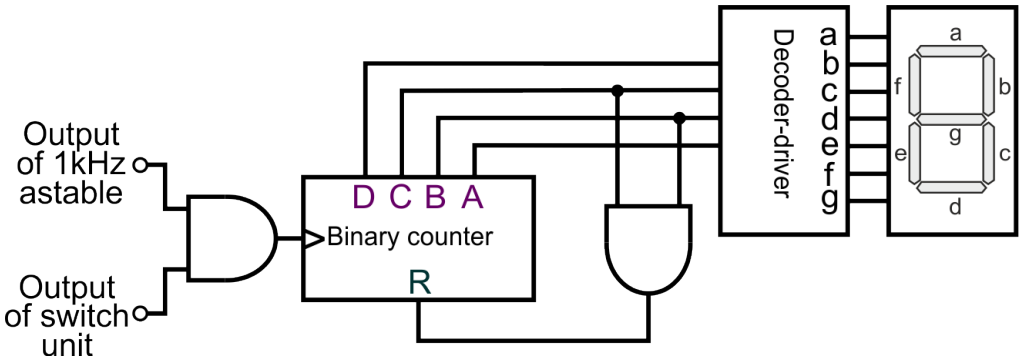
An electronic game requires 'dice' that can generate numbers from 0_{10} to 5_{10}.
Design a counting system with the following features:

- a switch is pressed and held down to make the 'dice' roll;
- when it is released, a number between 0_{10} and 5_{10} is displayed on a seven-segment display.

The solution is a counting system that:

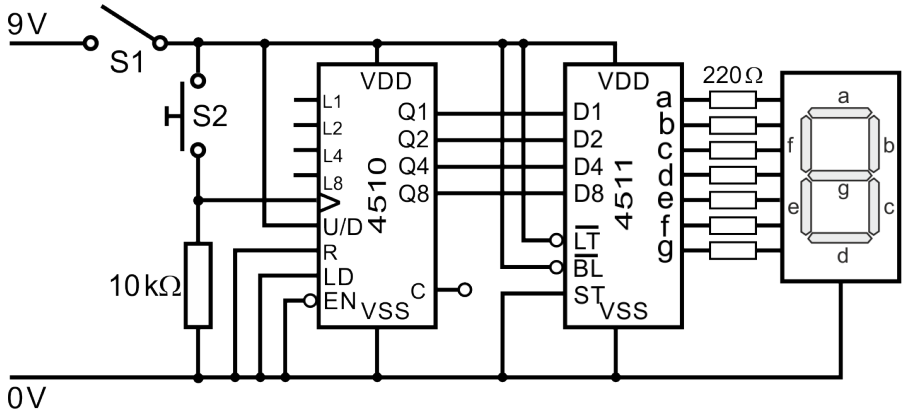
- resets the counter on the number '6_{10}';
- uses a switch unit that outputs logic **1** when the switch is pressed;
- uses an astable with a frequency too high for the human user to follow on the display.

Part of the solution, showing the reset circuit, is shown below:



Investigation 3.3

1. Set up the following 1-digit counting system on your simulation program.



a) Ensure that the voltage setting for the ICs is the same as the power supply.

b) Close switch **S1** and then press and release switch **S2** about twenty times.
Comment on what you observe.
.....
.....

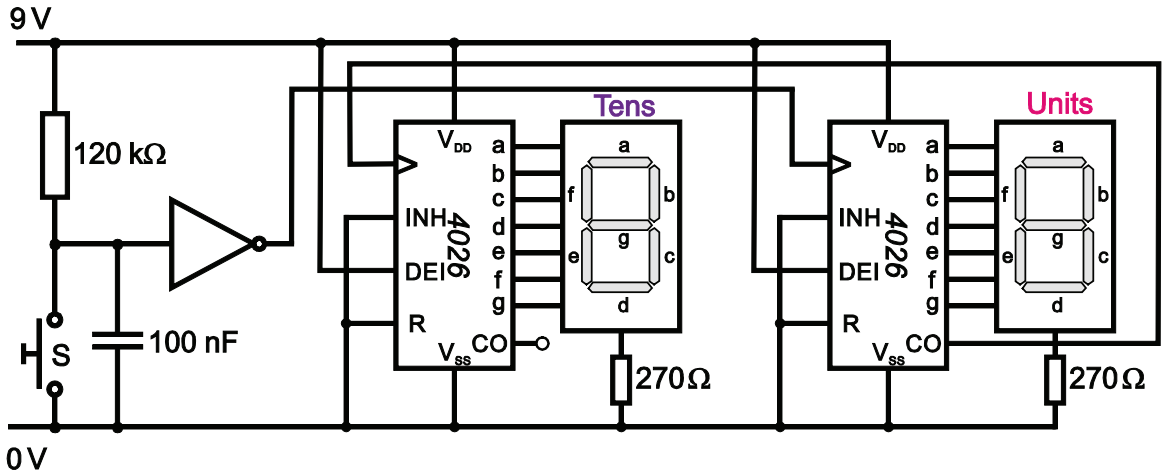
c) Investigate the effect of connecting the U/D pin of the 4510 IC to 0 V rather than 9 V.
.....
.....

d) There is a possible flaw in the design of the counting system. Can you predict what it is?
.....
.....

e) If you are using Circuit Wizard go to Project then Simulation and click on 'bounce'.
Repeat part a) and comment on what you observe.
.....
.....

f) Design a de-bounce sub-system based on a NOT gate monostable to give a delay of about 10 ms and add it to the clock input.
Show your final design in the space below.

2. Set up the following two-digit counting system on your simulation program. Check that 'switch bounce' is activated.



a) Press and release the push switch a large number of times.
Comment on how well the system works.

.....

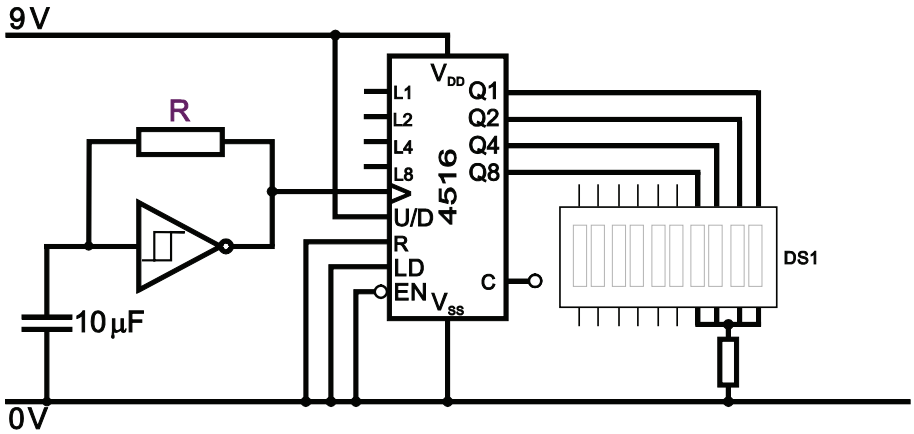
.....

b) Replace the 120 kΩ resistor with a 12 kΩ resistor and observe the effect.

.....

.....

3. Set up the following binary counting system.

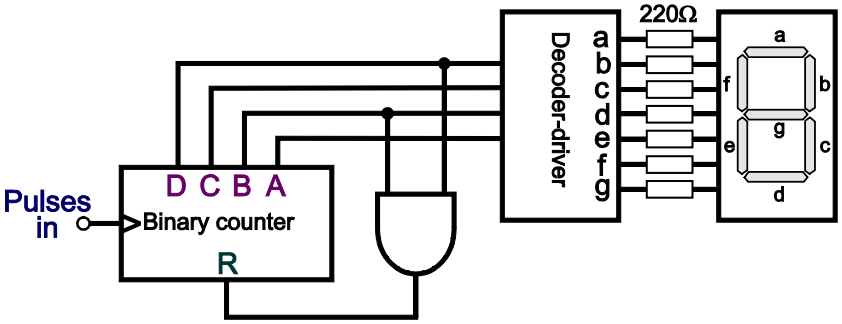


a) Calculate a suitable value for resistor R to provide a clock frequency of 0.5 Hz.
b) Study the output pattern on the bar graph display to confirm that it produces a binary count.
c) Connect the U/D pin to 0 V and confirm that it produces a binary down count.

Exercise 3.7

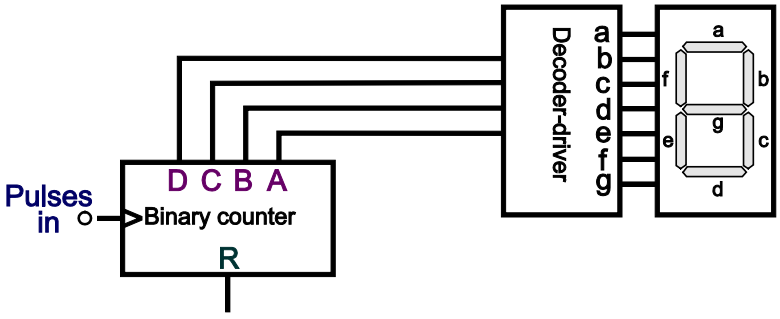
1. The following circuit shows a binary counter, connected to a decoder-driver and seven-segment display. The clock input is pulsed ten times.

Complete the table to show the resulting sequence shown on the seven-segment display.



Clock Pulse	Display
0	0
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

2. Complete the following circuit to ensure that the largest number displayed is '**7**'.

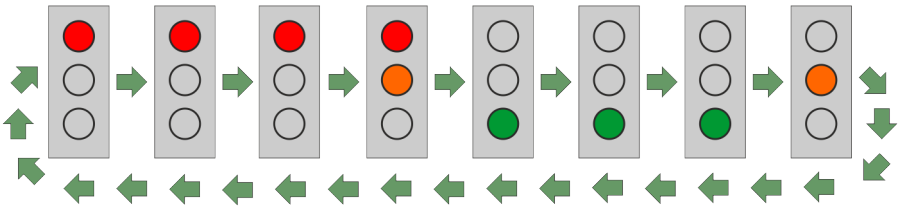


Counters as Sequence Generators

A combinational logic system attached to the outputs of a counter IC can be used to generate a control sequence. This will be demonstrated by working through an example below.

Sample design

A traffic light control system must generate the following lighting sequence:

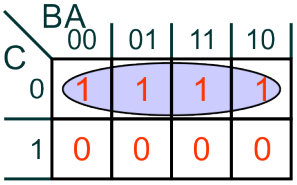


There are eight steps in the sequence. This requires three counter output bits ($2^3 = 8$).
The truth table for this sequence follows:

Inputs			Output		
C	B	A	Red	Amber	Green
0	0	0	1	0	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	0	1	0

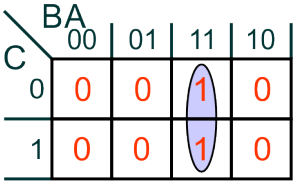
Using Karnaugh maps to determine the Boolean expression for each output gives:

For **Red**:



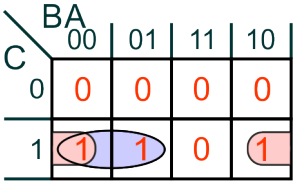
Red = $\overline{C}$

For **Amber**:



Amber = **B**.**A**

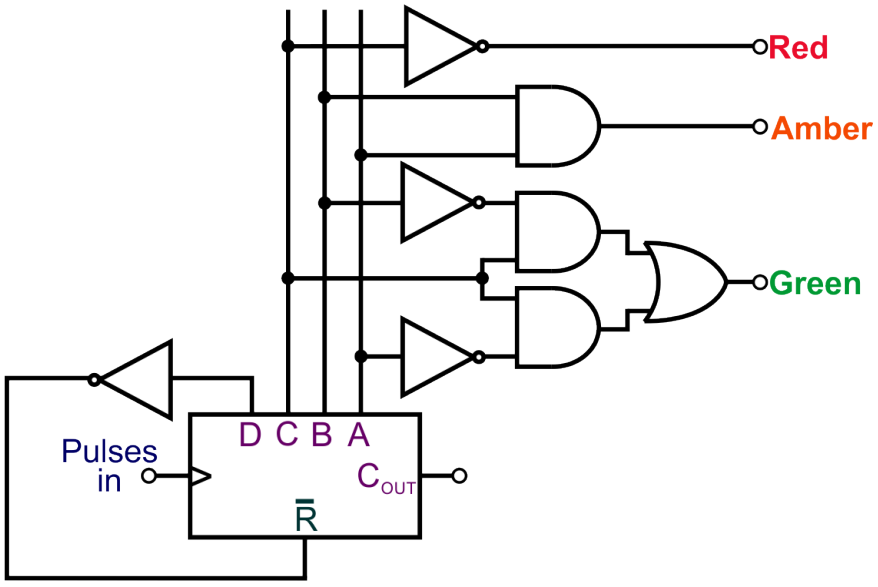
For **Green**:



Green = $\overline{B}.C + \overline{A}.C$

(Some of these results could have been obtained from inspection of the truth table.)

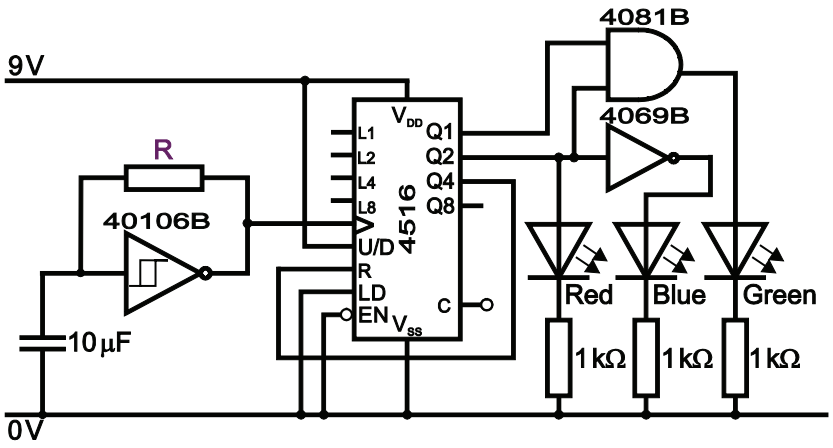
The logic system can now be constructed and added to the counter outputs.



The counter resets as soon as output **D** goes to logic **1**, i.e. on the eighth pulse. The sequence then repeats.

Investigation 3.4

1. Set up the following sequence generator.
Output Q1 is the least significant bit.

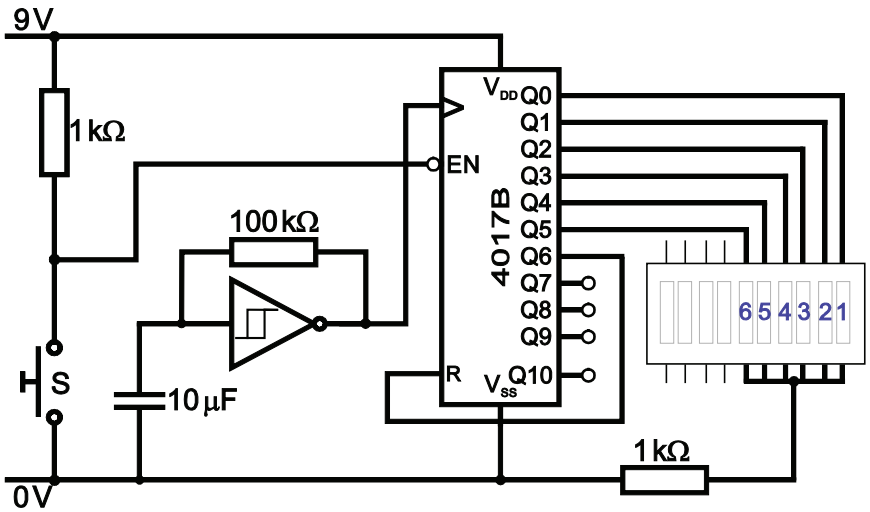


a) Analyse the circuit diagram and predict the sequence. Record your prediction in the truth table below

Counter outputs		LEDs		
Q2	Q1	Red	Blue	Green
0	0			
0	1			
1	0			
1	1			

b) Replace the 10 kΩ resistor with a 200 kΩ resistor to slow down the sequence to check your prediction

2. Set up the following circuit – a design for an electronic dice.



a) Close switch **S** for several seconds to confirm that outputs '1' to '6' come on in turn.

b) What is the flaw in the design of this electronic dice?

.....

.....

c) Replace the 10 µF capacitor with a 10 pF one and retest the dice.
It should now be less predictable.

d) Modify the circuit by connecting the Reset pin to 0 V and connecting the display to various counter outputs to make different light sequences.
Experiment with different values of resistor and capacitor to change the pulse frequency to improve the visual effect.

Exercise 3.8

A disco light sequence for a group of four lamps is shown below.

Lamp 1	Lamp 2	Lamp 3	Lamp 4
Off	On	On	On
On	Off	On	Off
Off	On	On	Off
On	Off	Off	Off
Off	On	Off	On
On	Off	Off	Off
Off	On	On	Off
On	Off	Off	Off

a) Complete the truth table for the sequence, assuming that the lights are active-high:

Inputs			Outputs			
C	**B**	**A**	**Lamp 1**	**Lamp 2**	**Lamp 3**	**Lamp 4**
0	0	0	**0**	**1**	**1**	**1**
0	0	1				
0	1	0				
0	1	1				
1	0	0				
1	0	1				
1	1	0				
1	1	1				

b) Complete Karnaugh maps for each lamp, and hence derive the Boolean expressions:

Lamp 1:

C \ BA	00	01	11	10
0				
1				

Lamp 1 = ...

Lamp 2:

C \ BA	00	01	11	10
0				
1				

Lamp 2 = ...

Lamp 3:

C \ BA	00	01	11	10
0				
1				

Lamp 3 = ...

Lamp 4:

C \ BA	00	01	11	10
0				
1				

Lamp 4 = ...

c) Design the logic system and connect to counter.

Add any connections needed to reset the counter.